BODO JOHANNES RÜCKAUER

# EVENT-BASED VISION PROCESSING IN DEEP NEURAL NETWORKS

# Event-Based Vision Processing in Deep Neural Networks

---

**DISSERTATION**

zur

Erlangung der naturwissenschaftlichen Doktorwürde
(Dr. sc. nat. )

vorgelegt der

Mathematisch-naturwissenschaftlichen Fakultät

der

Universität Zürich

von

**BODO JOHANNES RÜCKAUER**

von

Zürich ZH

**PROMOTIONSKOMMISSION**

Prof. Dr. Shih-Chii Liu
(Vorsitz, Leitung der Dissertation)
Prof. Dr. Tobi Delbruck
Prof. Dr. Giacomo Indiveri
Prof. Dr. Emre Neftci

Zürich, 2020

For my family.

# ABSTRACT

Over the past three decades, the field of neuromorphic engineering has produced sensors and processors that show great promise as efficient, brain-inspired systems. In parallel to this development, tremendous advances in Deep Learning (DL) have supplied highly accurate algorithms for computer vision. Unfortunately, these algorithms are not directly compatible with neuromorphic hardware. The present work bridges this gap by developing algorithms that leverage the power of Deep Learning while being suited for operation on neuro-inspired hardware.

The Dynamic Vision Sensor (DVS), a neuromorphic sensor used in this thesis, differs radically from conventional cameras by producing a stream of asynchronous pixel-events rather than regularly spaced light-intensity frames. These events signal changes in local brightness at high temporal resolution and wide dynamic range, which makes the sensor well suited for applications with spatio-temporal redundancy, difficult lighting conditions, or fast reaction time. However, the event-based nature of the sensor output impedes the application of standard computer vision techniques like Deep Neural Networks (DNNs).

In this thesis we demonstrate that DNNs can be adapted to operate in an event-based fashion, similar to how neural networks in the brain use discrete spikes for signal transmission. By converting Artificial Neural Networks (ANNs) trained with DL into Spiking Neural Networks (SNNs), we achieve some of the largest and most accurate spiking models for object classification to date. While information in SNNs for computer vision tasks is often encoded in the form of firing rates, the nervous system is known to employ other spike codes optimized for the requirements of a particular sensory pathway. Motivated by evidence of visual processing in humans occurring within milliseconds, we explore encoding schemes that make use of the precise timing of individual spikes to represent information. We show that high classification accuracy can be achieved in artificial systems based on few spikes per neuron.

Part of the widespread success of DL can be attributed to the ease with which algorithms and hardware are made accessible today. A beginner can readily find an online notebook that enables them to build, train, and run a full-scale DNN on a remote GPU within minutes, without any overhead setting up software and hardware. To achieve the same on a neuromorphic platform requires intricate understanding of the underlying hardware constraints, and a great deal of manual, low-level programming. One theme in this thesis is the reduction of these obstacles by providing automated tools to convert DNNs to the spike-domain, and to deploy them on neuromorphic hardware. Here, we mainly consider the Intel research processor Loihi, for which we developed a DNN compilation framework. Much of the previous work on SNNs is confined to simulations on general-purpose hardware, which allow no reliable characterization of the actual latency and power consumption of SNNs on dedicated hardware. By means of the toolchain developed here, we are able to perform such benchmarking on standard tasks from computer vision.

Though SNNs operate on spike events internally, they may receive conventional image frames as input. A more consistent approach is to use event-based input, e. g. from a DVS. In this work we discuss some of the benefits and challenges one can expect when thus combining event-based sensing and processing. In applications ranging from data compression to optical flow and Spiking Neural Networks, we demonstrate computational savings when operating on sparse, informative events rather than dense, redundant frames. Finally, we turn to a biological vision system, the retina, and show that a Dynamic Vision Sensor can be used to drive mouse Retinal Ganglion Cells in vitro - thereby opening a door for applications in retinal prostheses.

One recurrent theme in this thesis is the reduction of computational cost of neural networks. In a final study we ask whether the principle of sparse, event-driven updating can be transferred to standard ANNs without the use of spiking neurons. Inspired by how the DVS removes spatio-temporal redundancy from video, we apply a dynamic masking scheme to the layers of a DNN to reduce the number of operations during inference. The algorithm is shown to produce equivalent accuracy results at reduced computational cost on a range of vision tasks including human pose estimation and object detection in static and dynamic scenes.

This thesis contributes overall to a fruitful exchange between conventional computer vision on one hand and neuromorphic sensors and processors on the other. Both fields, to various degrees, share the motive for ever increasing efficiency, and many of the seeming restrictions of dedicated hardware, like reduced numeric precision, turn out to be desirable from an algorithmic perspective. Ultimately, we cherish the hope that in building massively constrained neuromorphic systems, we will one day understand more clearly how our brain accomplishes its tasks within a minimal space and energy budget.

# ACKNOWLEDGMENTS

I am grateful for the invaluable guidance of my advisor, Shih-Chii Liu. I especially appreciate her support in our external collaborations. Tobi Delbruck impressed on me that presenting science is about telling stories, him being one of the most entertaining story tellers I know. Giacomo Indiveri, who radiates encouragement, I thank for his example as a scientist and leader. Michael Pfeiffer was my advisor for the beginning of my PhD, and set the tone for my approach to solving challenges, integrating recent developments in the literature, and developing independent ideas.

A big thank you goes to the Admin team at INI for their world-class professionalism mixed with heartfelt human warmth. They supported me not only in the many day-to-day matters but went out of their way to help make possible the collaborations with Huawei, Intel, and Chungbuk National University.

I am grateful for the internship with Mike Davies and my "manager" Andreas Wild at Intel, who never made me feel managed, but who gave the sense my work was just as important to him as his own. Frighteningly smart and hard-working, he redefined what I considered unsolvable challenges. To Yong-Sook Goo and Jungryul Ahn at Chungbuk National University, a heartfelt thanks for their friendship and superb hospitality to my family and me during our three months stay in South Korea. They broadened my horizon both on a scientific and a cultural level. Thanks to Alexander Simak at Huawei for his friendly collaboration, the invitation to visit his group in Moscow, and the patience to see all the administrative hurdles go through.

I am indebted to Daniel Neil: His ANN-SNN conversion script formed the basis for the SNN toolbox. I am grateful for the work with Edward Jones and Simon Davidson from the APT group at U. Manchester, with whom we developed an interface from the SNN toolbox to SpiNNaker. I likewise enjoyed the collaboration with Evangelos Stromatias at U. Seville, who visited Zurich to integrate their Megasim with our SNN toolbox. I wish to thank Josep Maria Margarit Taulé for the pleasant collaboration on the fascinating topic of neuromorphic beverage tasting.

I am deeply grateful to my parents who let me grow up in an environment that encouraged love for any form of knowledge without diminishing the value of family and service. Janine: You have been a constant source of strength and motivation to me, I could not have achieved what I did without you. Kealani and you are my greatest joy.

# CONTENTS

# ACRONYMS

| | |
|---|---|
| AEDAT | Address Event DATa |
| ANN | Artificial Neural Network |
| APS | Active Pixel Sensor |
| ATIS | Asynchronous Time-based Image Sensor |
| BN | Batch-Normalization |
| CIFAR | Canadian Institute For Advanced Research |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| DAVIS | Dynamic And Active Pixel Sensor |
| DAE | Denoising AutoEncoder |
| DL | Deep Learning |
| DNN | Deep Neural Network |
| DRAM | Dynamic Random Access Memory |
| DV | Dynamic Vision |
| DVS | Dynamic Vision Sensor |
| EDP | Energy Delay Product |
| Flop | Floating-point operation |
| FoM | Figure of Merit |
| FPGA | Field-Programmable Gate Array |
| fps | frames per second |
| GPU | Graphics Processing Unit |
| GUI | Graphical User Interface |
| IF | Integrate-and-Fire |
| INRC | Intel Neuromorphic Research Community |
| KITTI | Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago |
| KLD | Kullback-Leibler Divergence |
| LK | Lucas-Kanade |
| LSB | Least-Significant Bit |
| LSTM | Long Short Term Memory |
| MAC | Multiply-ACcumulate |
| mAP | mean Average Precision |

| | |
|---|---|
| MEA | MicroElectrode Array |
| MNIST | Modified National Institute of Standards and Technology |
| MSB | Most-Significant Bit |
| MSE | Mean Square Error |
| Op | Operation |
| PSP | Post-Synaptic Potential |
| PSTH | Peri-Stimulus Time Histogram |
| ReLU | Rectifying Linear Unit |
| RGC | Retinal Ganglion Cell |
| RNN | Recurrent Neural Network |
| SNN | Spiking Neural Network |
| SNN TB | SNN toolbox |
| STDP | Spike-Timing Dependent Plasticity |
| SRAM | Static Random Access Memory |
| STG | STimulus Generator |
| TPC | Temporal Pattern Code |
| TTFS | Time-To-First-Spike |
| TTL | Transistor-Transistor Logic |
| USB | Universal Serial Bus |
| YOLO | You Only Look Once |

*Human subtlety will never devise an invention*
*more beautiful, more simple or more direct*
*than does nature because in her inventions*
*nothing is lacking, and nothing is superfluous.*

— Leonardo da Vinci

# 1 INTRODUCTION

Engineers at all times have looked to nature for inspiration - when searching for the most efficient wind turbine, the most streamlined train design, the ship coating with the least drag. Some - Leonardo da Vinci's flying machine no less - were never successful. With the brain being regarded by some as the most complex system in the universe (Kaku, 2014), the task taken on by *neuromorphic engineering* is daunting: to understand neural computation by building it in silicon. The present work contributes to this effort by investigating the mechanisms of visual processing with retina-like vision sensors and Artificial Neural Networks (ANNs).

Visual processing in artificial and biological systems can be approached from three directions (Chance *et al.*, 2020): **Function** (what does it do?), **algorithm** (how does it work?), and **hardware** (how is it built?). This thesis touches on all three aspects, the main focus lying on algorithms. The visual functions considered here range from object detection and classification in images to pose estimation in videos. The algorithms in this thesis revolve mostly around SNNs, with an excursion to a network compression method for video processing. Most of our algorithms were initially run in simulation on general-purpose hardware, but some of the SNNs were later ported to Intel's neuromorphic chip Loihi (Davies *et al.*, 2018). Also in terms of hardware, this work makes repeated use of the DVS (Lichtsteiner *et al.*, 2008), a camera that mimics the early stages of visual processing in the retina. To better understand the contribution of this work (Sec. 1.2) along these three research vectors, we will briefly summarize the recent developments of the field (Sec. 1.1), and introduce some basic terminology. The chapter closes with a detailed review of SNNs (Sec. 1.3), which form the core of this work.

## 1.1 CONTEXT

### 1.1.1 Deep Learning and Neural Networks

To solve the vision tasks considered in this thesis, we employ Deep Learning (DL) (LeCun *et al.*, 2015), the main driver of advances in computer vision over the past decade. One particular type of DL models is represented by DNNs, a layered collection of inter-connected processing elements, neurons. These neurons and their connections (synapses) are inspired by the nervous system (Fukushima, 1980;

Fukushima *et al.*, 1988; Riesenhuber *et al.*, 1999; Rosenblatt, 1962) but discard most of its biological detail, in particular with respect to neuronal dynamics, plasticity, and recurrent connectivity. The degree to which artificial systems can benefit from increased biological detail is subject to ongoing debate (Chance *et al.*, 2020; Cox *et al.*, 2014). Simplistic neuron models are easier to scale up and run efficiently on standard hardware, which led to state-of-the-art results in many machine learning benchmarks (e. g. Krizhevsky *et al.*, 2012; Szegedy *et al.*, 2017), sometimes surpassing human performance (Esteva *et al.*, 2017; He *et al.*, 2015). However, this "brute force" approach seems to reach its limits in some tasks that come natural to animals, like few-shot learning (being able to distinguish new objects after seeing only one or a hand-full of examples) (Riesenhuber *et al.*, 1999), incremental learning (acquiring new skills without losing previous knowledge) (Kirkpatrick *et al.*, 2017), or learning in the absence of teacher signals (self-, or semi-supervised learning). It is in these tasks where adding biological features (like cortical structure and circuitry, attention mechanisms, and memory consolidation during sleep) has led to promising results (Kemker *et al.*, 2018; Kubilius *et al.*, 2019; Masse *et al.*, 2018; Rikhye *et al.*, 2018).

This thesis takes a step towards increased biological realism in ANNs by equipping the cells with attributes of spiking neurons (Maass, 1997b), e. g. a state variable (the membrane potential) and digital signal transmission (the action potential) triggered by a voltage threshold crossing. These attributes are at the same time the blessing and the curse of SNNs: 1) Statefulness enables temporal integration similar to Recurrent Neural Networks (RNNs) (Kim *et al.*, 2019a; Kugele *et al.*, 2020; Pozzi *et al.*, 2018), but implies a substantial increase in memory footprint. 2) Spike-based operation is beneficial because it replaces costly Multiply-ACcumulate (MAC) operations with simple additions. On the other hand, encoding an analog value (as present in the ANN) by one or more digital spikes is likely to result in discretization errors and may add up to a higher total operation count (c. f. Sec. 2.1). 3) A threshold on the membrane potential leads to spatio-temporal sparsification, but is one of the main obstacles in training SNNs (c. f. Sec. 1.1.2). 4) As computation in the SNN occurs asynchronously due to spike events, the latency to get a first rough output may be quite short - a phenomenon termed *pseudo-simultaneity* (Perez-Carrasco *et al.*, 2013). However, asynchronously reading and writing states from and to external memory weighs heavy on the power consumption.

Overall, SNNs can be beneficial because of cheaper elementary operations, the exploitation of sparsity, and the algorithmic advantage of having a state variable. But to exploit these benefits one has to address the difficulty of training the SNN and running it efficiently in view of the increased and unpredictable memory traffic. Current approaches to these challenges will be introduced below.

### 1.1.2 Training and Conversion of Spiking Neural Networks

One reason why added biological realism in DL may be considered an obstacle rather than a feature is the prevalent mode of training using a form of error back-propagation (Rumelhart *et al.*, 1986). In backpropagation, one defines a loss function that computes the deviation of the network output from a target output. The network parameters (the connection strength between neurons) can be adjusted to

minimize the overall loss on a set of training samples, following a gradient descent along the loss surface towards a local minimum. The contribution of each parameter to the total loss (*credit assignment*) is determined using the chain-rule when minimizing the loss function with respect to the network parameters. This optimization requires all computations within the network to be differentiable, which is the case in standard ANNs. In SNNs[1] however, the spike generation mechanism SNNs typically involves a threshold operation, which has no derivative. Thus, training SNNs with backpropagation is not possible without modifications to the way we compute the weight update.

Solutions to the problem of training SNNs can again be found by looking towards nature, where synapses display plasticity based on local error signals, involving only the activity of neighboring cells (Gerstner *et al.*, 1996; Gerstner *et al.*, 1993; Hebb, 1949). Such Hebbian or Spike-Timing Dependent Plasticity (STDP)-based learning rules have been successfully applied to DNNs (Brader *et al.*, 2007; Diehl *et al.*, 2015a; Gerstner *et al.*, 2018; Gütig *et al.*, 2006; Kheradpisheh *et al.*, 2018; Masquelier *et al.*, 2007; Pehlevan, 2019), but the lack of a supervised teaching signal has limited results to comparably simple tasks. A remedy has been developed by smoothing out the non-differentiable threshold operation in SNNs (Hunsberger *et al.*, 2016; Lee *et al.*, 2016) or by providing *surrogate gradients* (Neftci *et al.*, 2019; Shrestha *et al.*, 2018). These spike-based backprop methods achieve higher accuracy than SNNs trained locally, and reach better accuracy on event-based datasets than converted SNNs. However, they are typically more time- and resource-consuming, and are not readily applicable to some of the challenging computer vision benchmarks like ImageNet (Russakovsky *et al.*, 2015).

In this thesis we develop ANN to SNN conversion methods (Cao *et al.*, 2015; Diehl *et al.*, 2015b; Rueckauer *et al.*, 2017; Sengupta *et al.*, 2019; Zambrano *et al.*, 2019), which circumvent the problem of direct SNN training by transferring a backprop-trained ANN into the spike domain. This approach allows leveraging the power of DL tools to start off with a state-of-the-art model of arbitrary size. However, the transformation to an SNN usually involves approximation errors. The first chapter of this work is devoted to reducing this accuracy loss during conversion.

### 1.1.3 Neuromorphic Hardware

The algorithmic foundation for the success of DL (multilayer neural networks, convolutional layers, backpropagation, large datasets) had been in place many years before the explosion of DL around 2012 (choosing the success of AlexNet (Krizhevsky *et al.*, 2012) at the large-scale ImageNet competition (Russakovsky *et al.*, 2015) as milestone). Thus, one can argue that this extraordinary development was to a large extent enabled by advances in GPU hardware, which allowed training deeper models on more data in a shorter period of time[2]. We may be on a similar verge in the domain of neuromorphic computing. SNNs have been around for decades,

1 We use the term DNN to cover both the spiking sub-class of SNNs and the analog-valued sub-class of ANNs, as both types can be "deep".

2 When Yann LeCun achieved a breakthrough by training his LeNet model on MNIST handwritten digits in 1989, the training took 3 days to complete (LeCun *et al.*, 1989). Today, an off-the-shelf laptop does the same job in a few seconds.

and have recently been scaled up to millions of neurons in software simulations (Rueckauer *et al.*, 2017; Sengupta *et al.*, 2019). However, the execution of these deep networks on general-purpose hardware is many times slower than using ANNs. Research efforts in both industry and academia are supplying increasingly large hardware platforms (Basu *et al.*, 2018; Davies *et al.*, 2018; Furber *et al.*, 2014; Merolla *et al.*, 2014; Qiao *et al.*, 2015) that optimize the execution of spike-based algorithms in terms of speed and power consumption. Realizing that a major bottleneck in running stateful SNNs is the transfer of weights and neuron states from chip to external memory and *vice versa*, these architectures allocate memory on-chip in close proximity to the processing units, which contrasts the centralized memory layout in von-Neumann architectures. But since on-chip memory (typically Static Random Access Memory (SRAM)) is expensive in terms of area, these neuromorphic platforms impose restrictions on the numeric precision of neuron states and synaptic weights, as well as the number of supported cells and connections. Consequently, programming these chips involves a great deal more effort and low-level understanding of the underlying implementation, as compared to applying a DL tool on general-purpose hardware. In this thesis we present a compiler for deep SNNs on Intel's neuromorphic processor "Loihi", which facilitates rapid deployment and evaluation with an interface as easy to use as the DL framework Keras (Chollet, 2015).

### 1.1.4 Event-Based Sensors

Signal processing with brain-inspired hardware and algorithms may be considered incomplete unless combined with brain-inspired *sensing*. Standard cameras generate still images with analog pixel values, which require some form of encoding step to make them compatible with spike-based processing, e. g. by generating Poisson spike trains at rates proportional to the pixel values. In contrast, bio-inspired cameras (Liu *et al.*, 2015) like the DVS (Lichtsteiner *et al.*, 2008) or Asynchronous Time-based Image Sensor (ATIS) (Posch *et al.*, 2011) sample the scene dynamically at high temporal resolution, and report pixel-wise changes in log light intensity. The result is a stream of events with a pixel address, time stamp, and polarity (indicating the sign of brightness change), which formally resembles the spike response of ON and OFF Retinal Ganglion Cells (RGCs) in animal retinas. This distinct form of signal acquisition is not easily applicable to conventional frame-based computer vision algorithms (for a review of successful examples see Gallego *et al.*, 2019), but appears well suited for subsequent processing in Spiking Neural Networks (Amir *et al.*, 2017; Farabet *et al.*, 2012; Kugele *et al.*, 2020; Stromatias *et al.*, 2017; Yousefzadeh *et al.*, 2019). In this thesis we evaluate the use of event streams from the DVS in applications ranging from optical flow and neural networks to stimulation of mouse RGCs in vitro.

## 1.2 CONTRIBUTION AND STRUCTURE OF THESIS

### 1.2.1 Converting and Encoding Spiking Neural Networks

→ *Chapter 2*

When we began this work early 2016, the only standard benchmark in computer vision where deep SNNs were applied was the task of recognizing handwritten digits (MNIST, LeCun *et al.*, 1998). While this task is an important proof of concept, its value beyond a first baseline check is doubtful: Already in 2016 this task was considered solved (i. e. further improvements in accuracy were unlikely) and accuracies below 98% hardly acceptable. In addition, a high accuracy on MNIST is by no means a confident indication that the method will scale up to more challenging tasks based on natural images, due to the stereotypical properties of MNIST digits (grey-scale, high contrast and texture-free structure, uniform background). To remedy this situation, we set out to test whether the ANN-to-SNN conversion method developed in our group (Diehl *et al.*, 2015b) would be able to achieve satisfying performance on some of the more difficult benchmarks of computer vision, in particular CIFAR-10 (Krizhevsky, 2009) and ImageNet (Russakovsky *et al.*, 2015). This research led to a series of improvements, including a more robust method to map neuronal activity into the right dynamic range, the inclusion of biases in the neuron model, a noise-free input encoding, a spike-based implementation of common ANN features such as MaxPooling and Softmax, and a theoretical derivation of the spikerate dynamics in higher layers, which motivated the use of a loss-less reset mechanism of the membrane potential after spike generation (Rueckauer *et al.*, 2016b). One of the objectives in this endeavor was the ability to take an arbitrary pre-trained ANN as-is, without the need to apply any kind of restrictions to the architecture, neuron model or training procedure, and run it as an SNN. In this way we ultimately succeeded in converting the state-of-the-art Inception-V3 architecture with little loss in accuracy on the ImageNet dataset (c. f. Sec. 2.1 and Rueckauer *et al.*, 2017). In an effort to facilitate the process of transferring models to the spiking domain and to evaluate them efficiently, we developed a toolbox for SNNs[3] that automates the conversion and simulation workflow. At that time, the DL ecosystem was evolving rapidly with no framework clearly dominating. To support this diverse range of models, we equipped the SNN toolbox with an extensible frontend for the most common ANN training libraries. Likewise, the toolbox provided a backend for many common SNN simulation libraries. Since the initial release of the software, front- and backend have been kept up-to-date with recent developments in both the DL and the neuromorphic community, enabling for instance deployment of converted SNNs on the SpiNNaker and Loihi platforms by Patiño-Saucedo *et al.*, 2020 and Massa *et al.*, 2020, respectively.[4]

Despite the high accuracy that could be achieved on full-scale DNNs using the conversion approach described above, the run-time cost of the resulting SNNs was often prohibitive. Approximation errors in higher layers due to the discrete nature of spike processing could only reduced at long simulation times. Since the neuron model employed a rate-code to represent signals, the large amount of spikes that

3 https://snntoolbox.readthedocs.io/, accessed Aug 2020.

4 At the time of writing, the toolbox has been forked 64 times, received 140 stars on github, and has been downloaded over 17000 times.

needed to be processed led to an increased operational cost compared to the ANN (c. f. Sec. 2.1 and Rueckauer *et al.*, 2017). While other groups improved the efficiency of rate-based models e. g. by endowing their neuron model with an adaptation mechanism (Zambrano *et al.*, 2016), we explored temporal codes where information is contained within the exact timing of individual spikes. In a first attempt, we went to the extreme of allowing only a single spike per neuron, with the inverse latency of that spike representing the instantaneous firing rate. This method passed the MNIST test with a minimal amount of spikes (c. f. Sec. 2.2 and Rueckauer *et al.*, 2018), but showed unacceptable accuracy loss on CIFAR-10 because of this encoding's sensitivity to possibly missing long-latency spikes. In our search for a more robust but still efficient spike code we found a compromise in an encoding scheme based on temporal spike patterns. This encoding guarantees correct approximation of the ANN within a simulation duration generally shorter than the rate-coded SNN, and using fewer operations due to the reduced number of spikes. The method was applied successfully to MobileNet (Howard *et al.*, 2017) on ImageNet (c. f. Sec. 2.3).

The exploration of these spike encodings - rate, latency, and pattern - sheds light on the question in what circumstances it may be advantageous to employ spikes for processing a DNN, and which spike code to use then.

### 1.2.2 Processing SNNs on Neuromorphic Hardware

→ *Chapter 3*

The SNN conversion results above, as well as most related studies in the literature, suffer from one significant deficiency: The underlying methods are evaluated in software, and the reported performance metrics typically consider only the number of spikes or required operations to achieve a certain accuracy. Unfortunately, these computation-based metrics do not translate well to actual run-time and power consumption in hardware, where the cost of memory traffic exceeds computation. Rigorous benchmarking results of deep SNNs on neuromorphic hardware are still rare (Amir *et al.*, 2017; Esser *et al.*, 2016; Orchard *et al.*, 2015b; Serrano-Gotarredona *et al.*, 2015; Stromatias *et al.*, 2013) but are essential for gauging the performance of SNNs under realistic conditions. One factor impeding such studies is the large overhead required to deploy a DNNs on special-purpose hardware. To facilitate this process, we developed a compiler for DNNs on Loihi, which shields the user from all the low-level details and architectural constraints of the underlying hardware, and provides a familiar and easy-to-use interface based on Tensorflow / Keras. This compiler solves an optimization problem of distributing the neurons within a neural network across the neurocores of Loihi while making optimal use of the limited resources available. This compiler is integrated with the SNN toolbox described above, thus enabling end-to-end deployment of pre-trained ANNs on neuromorphic hardware. We applied this workflow to the MobileNet architecture on CIFAR-10 and achieved the highest accuracy reported on this dataset on neuromorphic hardware.

### 1.2.3 Event-Based Sensing

→ *Chapter 4*

The majority of the spike-based processing methods described above and in the literature are evaluated on common frame-based datasets from computer vision,

which is in part due to the (slowly diminishing) lack of suitable event-based datasets, and in part due to the aim to position the work within the wider context of DL. It is doubtful that this stretch between still images and spike-based processing should be ideal (Deng *et al.*, 2020). We begin Chapter 4 by providing some motivating examples of using event-based *sensing* in conjunction with event-based *processing*, e. g. in the computation of optical flow (Sec. 4.1.2, c. f. Liu *et al.*, 2019). We then show that this argument can be extended to SNNs, by injecting DVS events directly into a converted SNN for hand gesture recognition and robot tracking (Sec. 4.2, c. f. Rueckauer *et al.*, 2019a). The chapter concludes with a feasibility study of using event-sensors to drive RGCs in the context of retinal prostheses. We develop an interface between the DVS and the STimulus Generator (STG) of a MicroElectrode Array (MEA) to inject current pulses into mouse retina tissue in vitro, and characterize the stimulus properties to elicit cell responses at highest temporal resolution (Sec. 4.3). This study is a step towards utilizing bio-inspired technology in bio-medical applications.

### 1.2.4 Network Contraction for Efficient Video Processing

→ *Chapter 5*

The event stream coming from DVS cameras is spatially highly informative as only *changes* in brightness are perceived. In contrast, a frame-based video is often characterized by high spatio-temporal redundancy, and a naive processing therefore wasteful. Over the backdrop of event-based sensing and processing, we developed a method that dynamically masks feature maps in neural networks. The basic idea is to replace the Rectifying Linear Unit (ReLU) activation function with a binary mask tensor, which transforms the network graph into a series of matrix-vector products. This chain of products can be contracted into a single matrix. The resulting linearized graph will be an accurate compressed version of the original network until changes in the input scene require an update of the feature masks. We demonstrated applicability of this compression scheme on object detection, digit denoising, car tracking and human pose estimation.

### Note on Published Material Included in this Thesis

Several of the chapters in this thesis contain material that has been published or is currently under review, as indicated by a footnote at the beginning of each chapter. All content is included with the authorization of the respective publisher, and the consent of the co-authors. For all included papers I am the first author and main contributor.

## 1.3 REVIEW OF SPIKING NEURAL NETWORKS

Since a large part of this thesis is based on SNNs, we attempt to give an overview over developments in the field over the past six years. As there have been roughly 100 publications on the subject within this time frame, we focus on questions most closely related to the present work: What methods have been developed to improve conversion of ANNs into SNNs? Which spike encodings show the most

promise? How has the operational cost been reduced? How have SNNs been applied in conjunction with neuromorphic hardware and event cameras? We do not cover recurrent architectures here, nor spike-based training (some references are given in Sec. 1.1 above). For a recent review see also Pfeiffer *et al.*, 2018.

### 1.3.1 Reducing Operations

Our work was among the first to explicitly demonstrate that SNNs are able to solve tasks as difficult as ImageNet with satisfying accuracy, albeit at a high number of computations due to the underlying rate-code. Since then, much research effort has been directed towards making SNNs more efficient. These approaches can be roughly categorized into methods applied during ANN training, and methods that modify the SNN neuron model or processing mode.

Falling into the first category, Neil *et al.*, 2016 demonstrate up to 70% reduction in operations using simple techniques such as Dropout (to teach network to deal with missing spikes), activation regularization with L2 norm (to reduce overall spike rates, i. e. induce temporal sparsity) and L1 norm (for spatial sparsity). Such methods are certainly appropriate whenever training a model from scratch, but the reduction for larger models than MNIST has not yet been investigated rigorously.

Sorbaro *et al.*, 2020 applied a similar technique by including a loss term for synaptic operations during training to reduce overall network activity. The accuracy drop due to the forced silencing of certain neurons is taken into account by quantization-aware training. The reported training effort seems to be quite high with a schedule of 30 rounds of training the model for 350 epochs each, with increasingly strong synOp loss term. But as the training phase is typically a one-time effort, it may be worth the investment.

Patiño-Saucedo *et al.*, 2020, who use the SNN toolbox to deploy a model on SpiNNaker for MNIST and N-MNIST, likewise apply regularization on the network activities during ANN training and report a 34× spike reduction at less than 1% drop in accuracy.

Rathi *et al.*, 2020 present a hybrid approach between conversion and SNN training: They first take a converted SNN and use its weights and thresholds as an initialization step for spike-based backpropagation. In a second phase, they perform incremental spike-timing dependent backpropagation (STDB) on this carefully initialized network to obtain an SNN that requires fewer time steps during inference compared to purely converted SNNs.

As an example of the second group of efficient conversion methods (based on enhanced SNNs processing), Zambrano *et al.*, 2016 equip their neuron model with an adaptation mechanism. When presented with a sequence of images, neurons are highly active at the beginning of a new sample presentation and then quickly adapt to spike only rarely until a new sample is shown. This method trades off spikes against more expensive neuron updates, but a more detailed cost comparison will be required to determine the net savings in hardware. An extension of this promising approach was presented recently (Zambrano *et al.*, 2019).

Another method that modifies the neuron model to achieve computational savings was proposed by Yousefzadeh *et al.*, 2019. Rather than using Integrate-and-Fire (IF)

neurons, they apply a form of sigma-delta modulation together with quantization and a threshold on the deltas to reduce events. The quantization function displays hysteresis, which effectively filters out high-frequency fluctuations in the signal to be quantized. The results for MNIST fall in between our rate-coded and Time-To-First-Spike (TTFS)-coded models (a factor of $2\times$ less operations between each). By applying their method to the event-based DVS-Gestures dataset (Amir *et al.*, 2017), they exploit sparsity on three levels - spatial, temporal, structural (synaptic).

Yu *et al.*, 2020 likewise employs a sigma-delta encoding, using signed spikes. In addition, if the membrane potential $V$ exceeds the threshold $\theta$ by a multiple of the threshold, spikes have a magnitude of $\lfloor V/\theta \rfloor$, which speeds up convergence by a factor $3-4\times$ at iso-accuracy.

Several groups have evaluated temporal codes that use only a single spike (Mostafa, 2018; Rueckauer *et al.*, 2018; Stöckl *et al.*, 2019; Zhang *et al.*, 2018) or a limited number of spikes per neuron (Stöckl *et al.*, 2020; Zhang *et al.*, 2019, Sec. 2.3 in this thesis) rather than a rate to represent their activity. In the special case of a latency code, Kim *et al.*, 2018 assign different weights to spikes depending on the time delay within an interval to encode more information using fewer spikes. The single-spike approaches allow a maximal reduction in operations but have not yet been shown to scale up to real-world tasks. The few-spike approaches find a promising compromise in terms of operations and achieve high accuracy even on ImageNet, but take a step away from the asynchronous processing mode and biological realism of SNNs. These methods are discussed in more detail in Sections 2.2 and 2.3.

We believe that both the methods focusing on ANN training and the methods modifying the SNN model have their merit: The former are easy to implement on general-purpose hardware and result in a one-time cost; the latter are likely to bring higher savings but their more complex model induces an overhead when being implemented on dedicated hardware.

### 1.3.2 Improving Conversion

Compared to the challenges of direct spike-based training of SNNs, ANN conversion approaches may appear like a lazy shortcut. "There ain't no such thing as a free lunch", however, and conversion methods need to invest much of the work carried out by spike-based learning methods to prevent or mend the approximation errors that follow conversion.

As an example of trying to prevent conversion errors already during ANN training, Hunsberger *et al.*, 2015 apply Gaussian noise to the ANN to teach it robustness against spike variability, achieving state-of-the-art on CIFAR-10 at the time. They also use a leak parameter in their neuron model, which requires smoothing out the kink of the ReLU activation function during training. While the benefit of noise is evident from the results, it is less so for the leak.

An instance of post-conversion optimization was realized by Diehl *et al.*, 2015b, who observed spike rate saturation in their models. Such saturation (and, like-wise, under-representation) of rates is expected in a simulation setting with finite runtime and discrete time steps. To map the firing rates into the supported dynamic range,

they applied a weight normalization scheme to the converted model, which led to a new state-of-the-art on MNIST. This method was later extended by Rueckauer *et al.*, 2016b.

Chen *et al.*, 2018 address the problem of weight normalization (Diehl *et al.*, 2015b; Rueckauer *et al.*, 2017) at train-time, by clipping the ANN activations to $[0, 1]$. Further, they propose a mechanism to prevent erroneous spikes, by randomly sampling spikes at each neuron at run time. Their method results in fewer time steps until convergence, but these timesteps are more costly (which lacks a discussion).

Hu *et al.*, 2018 achieved a significant result by converting residual architectures (He *et al.*, 2016) with little loss in accuracy on CIFAR-10. A comparable network without skip connections dropped significantly in accuracy, though it was not made clear in which way skip connections are in fact beneficial in a spiking network. We hypothesize that skip connections may help stimulate neurons across all layers of the hierarchy at approximately the same time; we will see in Sec. 2.1.3.3 (resp. Rueckauer *et al.*, 2017) that the lack of synaptic input in higher layers may otherwise lead to bias-driven transients in the spike rates of these neurons.

Sengupta *et al.*, 2019 extended the conversion of ResNets to ImageNet. However, they still apply a large number of restrictions during ANN training (no MaxPooling, no biases, no batch normalization), and report a drop in accuracy of 4-5% at a $2\times$ increased number of operations compared to the ANN. Still, theirs was the first SNN work to test all 50000 samples of ImageNet.

The study by Kim *et al.*, 2019b stands out for being the first to perform large-scale object detection in the spike domain, using a "tinyYou Only Look Once (YOLO)" architecture (Redmon *et al.*, 2016). The conversion method itself differs little from the earlier ones (Rueckauer *et al.*, 2017).

The work by Stöckl *et al.*, 2019 can be seen as a form of population code rather than rate code: Each ANN neuron is replaced by $K$ spiking neurons[5], which all have fixed synaptic delays, and trained lateral connections, thresholds, and extra weight coefficients, for a total of 52-940 additional parameters per ANN neuron. This population code allows modelling arbitrary activation functions, not just ReLU - though spatial sparsity is lost if not using a rectifier. This approach is noteworthy for achieving the exceptionally high top-5 accuracy of 95.82% on ImageNet. However, it is doubtful that all this additional memory can be stored and fetched efficiently, considering the massive number of 3.11 Billion multi-parameter neurons in their ImageNet model.

Another approach to prevent conversion errors during training was presented by Wu *et al.*, 2020, who train the ANN in parallel with the SNN, using the spike count of the SNN as activation values in the ANN. This tandem learning framework offers the additional possibility of building in hardware constraints, such as limited weight precision and fan-in connections, to be progressively imposed during training.

### 1.3.3 Operating on Event-Based Vision Data

After initial studies on small event-based datasets (Farabet *et al.*, 2012; Perez-Carrasco *et al.*, 2013; Zhao *et al.*, 2015), notable efforts were made to provide

5 Typical values for $K$ range between 8 and 40.

benchmark datasets comparable to those used in computer vision. One approach was to synthesize event-datasets from still images (Hu *et al.*, 2016; Orchard *et al.*, 2015a). A caveat in using such "converted" datasets lies in the fact that a DVS recording events from frames has no way of capturing the real dynamics of the scene and bring in the temporal resolution of event-sensors. Thus, conclusions drawn from using these datasets cannot necessarily be extrapolated to real-world scenarios. More recently, datasets from natural scenes were recorded for gesture and object recognition (Amir *et al.*, 2017; Sironi *et al.*, 2018). An exhaustive list of event-based datasets covering a wide range of applications can be found online[6].

One of the challenges in using event-based vision datasets is the seeming incompatibility with conventional computer vision algorithms and ANNs. A common solution is to apply some form of post-processing stage on the event-stream to extract features that can be used as input to frame-based algorithms. One of the simplest but nevertheless successful approaches is to generate histograms from a constant number of events or from events collected over a constant time window. Extracting frames or features from events often requires heuristics to adapt to the given task or input statistics. In reviewing the related literature below, we look for studies that provide principled insights into this process. In addition, we are interested in methods that are able to operate on events directly, without a processing stage that possibly hides informative temporal characteristics of the event stream.

HOTS (Lagorce *et al.*, 2017) is a hierarchical architecture for pattern recognition similar to a Convolutional Neural Network (CNN) but with spatiotemporal filters that are learned in an unsupervised manner. Event streams coming from DVS cameras can be interpreted as time surfaces over the pixel address space, with characteristic features generated by object motion. HOTS matches a bank of templates to the spatio-temporal context of incoming events. Well-matching templates produce events themselves, which are processed in a similar way by subsequent stages in the hierarchy. The templates are learned by an initial clustering algorithm on the data. The HOTS approach is attractive because of the use of temporal information in the time surfaces rather than "flat" event frames. Drawbacks include the latency to generate the time surface, as well as the high computational cost of the clustering algorithm.

Sironi *et al.*, 2018 later extended the concept of time surfaces by introducing a memory of past events in a method called HATS. This memory allows a better treatment of noise events: Instead of simply taking the most recent events to produce a time surface as in Lagorce *et al.*, 2017, one can then average over a spatio-temporal neighborhood. Thanks to the increased robustness, using such a regularized time surface allows reducing the number of levels in the hierarchy, thus reducing the overall computational load.

As an example of a method that operates directly on the DVS events, Lee *et al.*, 2016 achieve the then best result on N-MNIST by training a single-layer fully-connected network on DVS events.

Rather than training a full model, Stromatias *et al.*, 2017 train a feature extractor SNN in an unsupervised fashion on the event stream. This SNN preprocessor transforms the DVS training data into a new frame-based training set in feature

6 https://github.com/uzh-rpg/event-based_vision_resources, accessed Aug. 2020

space, which captures the (non-Poisson) firing dynamics of DVS input neurons. This frame-based data is then used to train an ANN classifier, for instance a single fully-connected layer. After training, the resulting classifier weights can be deployed in a fully spiking network for inference. The method was successfully applied to a range of neuromorphic vision data sets, including Poker cards and MNIST variants. It represents an appealing way of dealing with irregularities in the DVS spike trains, which may otherwise become problematic in SNNs trained on binned event-frames (c. f. Sec. 4.2.3.2). A potential limitation of this method is given by the unsupervised feature extractor training.

The work of Amir *et al.*, 2017 contributes a new gesture recognition dataset captured with the DVS, and demonstrates high accuracy running on the TrueNorth chip. It may be argued that this task may be too easy as it already achieves 96.5% accuracy in their initial submission with a CNN constrained to ternary weights. The processing of the event stream is particular in that events are binned into frames using a temporal filter cascade, and then fed to the network in stacks of 6 frames, to provide temporal context. Such a frame stack is commonly seen in conventional video-recognition networks, e. g. when applying 3D convolutions on a large number of temporal input channels (Feichtenhofer *et al.*, 2016; Ji *et al.*, 2013; Karpathy *et al.*, 2014; Ng *et al.*, 2015).

Massa *et al.*, 2020 arrive at a similar conclusion when deploying their DVS-Gestures model on Loihi, using the SNN toolbox and compiler described in Chapter 3. The authors explore various ways of binning the events into frames for training. Their best solution uses 3 overlapping frames stacked together, with event polarity rectified.

With many of the methods discussed above still relying on heuristics and experimentation to suitably transform event streams, Afshar *et al.*, 2019 aim to provide a more thorough picture. They investigate event-based feature extraction using a range of spatio-temporal kernels with different surface decaying methods, decay functions, receptive field sizes, feature numbers, and back-end classifiers. The result are insights into performance vs complexity trade-offs, with a focus on hardware implementation. While this endeavor is highly relevant, one may question the extent to which we can expect generalization from the studied (toy) problem of falling airplanes with a single-layer-network.

A refreshingly new perspective for event-processing in SNNs is provided by Kugele *et al.*, 2020, who demonstrate the benefit of using delays in SNNs, together with skip connections, to achieve temporal integration across an event-stream. The processing mode is not fully asynchronous but somewhere between ANN and SNN. The experiments cover a wide range of neuromorphic datasets (N-MNIST, DVS-CIFAR-10, DVS-Gestures, N-Cars), where the method achieve accuracies on par with state of the art. Drawbacks include an increased number of hyperparameters, restrictions on the architecture to achieve a desired latency, and training with backpropagation-through-time.

An interesting view is provided by He *et al.*, 2020, who contrast SNNs with RNNs on neuromorphic data (N-MNIST, DVS-Gestures). Such a comparison is interesting because SNNs are stateful just as RNNs, and should do well on similar tasks that require temporal integration. Yet there has been little work in this area so far

(Bellec *et al.*, 2018; Kim *et al.*, 2019a; Nusselder, 2017; Pozzi *et al.*, 2018). The authors find that vanilla-RNNs perform as well as Long Short Term Memory (LSTM) and SNNs on N-MNIST, but have low accuracy on DVS-Gestures, which requires more temporal integration. At equal neuron count, LSTM does not achieve higher accuracy than SNNs on either task, which makes it doubtful that the additional number of parameters and operational cost is justified. Also, as input sparsity is increased, SNNs are able to maintain higher accuracy than RNNs or LSTM.[7]

To evaluate in which situations SNNs are beneficial, Deng *et al.*, 2020 compare ANNs, trained SNNs and converted SNNs on ANN-oriented workloads (MNIST, CIFAR), and SNN-oriented workloads (N-MNIST, CIFAR-DVS). The authors measure accuracy, number of operations, and the memory footprint. They find that on ANN-oriented workloads, SNNs can be advantageous over ANNs if the task is simple (MNIST, not CIFAR). On SNN-oriented workloads, the SNNs trained on spikes perform best in terms of accuracy and computations. While such a study is highly relevant, it unfortunately suffers from the same limitation as most related work, namely disregarding the different cost of memory traffic in ANNs vs SNNs. Further, the simple methods for training and conversion used in the study are likely suboptimal, i. e. one of the specialized algorithms discussed here might change the picture. We agree with the authors in their call for broader SNN benchmark datasets, different spike encoding schemes, and more meaningful performance statistics. The latter two aspects are taken up in Chapters 2 and 3, respectively.

### 1.3.4 Summary: State-of-the-Art of Deep SNNs

In an effort to gain an overview of the progress of vision processing in SNNs over the past, we collected more than 100 results published during the last seven years on the most common benchmarks. It became immediately clear that a detailed comparison is hindered by the lack of standardized performance metrics. For each reported result we tried to establish the number of neurons, parameters, ANN MAC operations, SNN synaptic operations and neuron updates, spike counts, number of simulation time steps, inference sample rate, and energy per sample. The only ubiquitous quantity was classification accuracy. Model size could in most cases also be inferred from the papers. We normalized the available performance metrics per dataset and computed a Figure of Merit (FoM) as the product of these normalized quantities, which was then used as marker size in Fig. 1.1. Given the sparseness of available performance data, this FoM has very little explanatory power. Nevertheless, some trends may be observed from plotting the accuracy over time for each dataset and algorithm type[8] (Fig. 1.1). Clearly, the field is active, with an acceleration in the number of publications, and increasingly more results on difficult tasks (in particular ImageNet), as well as event-based data. Simple datasets (MNIST variants) seem to have saturated early on. Results on neuromorphic hardware are still few

7 Input sparsity can be increased by reducing the time window over which events are accumulated into frames.

8 By categorizing the methods as *Local learning*, *Supervised spike-based training*, *Constrained training*, and *Conversion*, we loosely follow Pfeiffer *et al.*, 2018.

(diamond markers in Fig. 1.1). *Conversion* methods[9] dominate in numbers, and as of yet appear to give best results in terms of accuracy and cost. *Local learning* results were initially restricted to MNIST and showed inferior accuracy, but have recently been applied to N-MNIST and DVS-Gestures with state-of-the-art accuracy (Kaiser *et al.*, 2020). *Spike-based backprop* methods were also mostly applied to MNIST but recently also to some event-based datasets, which seem a more natural fit. While the overall development in terms of accuracy looks encouraging, it is difficult to judge whether any of the proposed methods constitute a breakthrough without getting a clearer picture of the associated cost.

9 *Constrained training* methods are a form of conversion where approximation errors are alleviated by training the original ANN with certain restrictions.

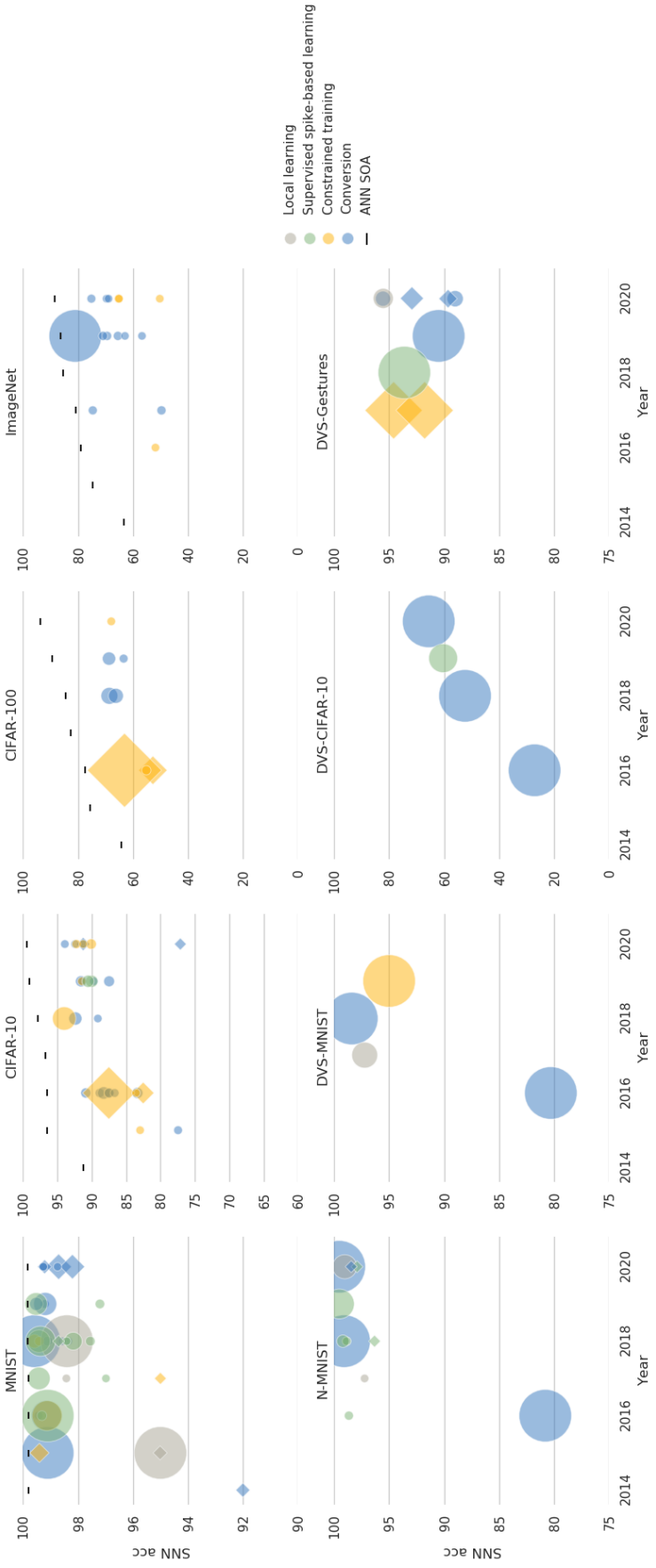


**Figure 1.1:** SNN results over the past seven years. Circular markers denote results in simulation on general-purpose hardware; diamond markers show results in dedicated neuromorphic hardware. The marker size indicates the performance metric (a combination of model size and operational cost). The upper row contains standard frame-based vision datasets; the bottom row contains neuromorphic event-based datasets. Black horizontal bars indicate state-of-the-art ANN results.

# 2 SPIKE ENCODING FOR CONVERSION OF DEEP NEURAL NETWORKS

In this chapter we explore methods to convert ANNs to SNNs using three different spike codes. Sec. 2.1 starts with the simplest and most common code based on firing rates. While effective at encoding SNNs of unprecedented depth, the redundancy inherent in a *rate code* leads to the development of temporal codes: A minimalistic *latency code* based on the timing of a single spike per neuron (Sec. 2.2), and a *pattern code* utilizing a short sequences of spikes with information encoded in individual spike times (Sec. 2.3).

## 2.1 FIRING-RATE CODE

SNNs can potentially offer an efficient way of doing inference because the neurons in the networks are sparsely activated and computations are event-driven. Previous work showed that simple continuous-valued deep CNNs can be converted into accurate spiking equivalents. These networks did not include certain common operations such as max-pooling, softmax, batch-normalization and Inception-modules. This work presents spiking equivalents of these operations therefore allowing conversion of nearly arbitrary CNN architectures. We show conversion of popular CNN architectures, including VGG-16 and Inception-v3, into SNNs that produce the best results reported to date on MNIST, CIFAR-10 and the challenging ImageNet dataset. SNNs can trade off classification error rate against the number of available operations whereas deep continuous-valued neural networks require a fixed number of operations to achieve their classification error rate. From the examples of LeNet for MNIST and BinaryNet for CIFAR-10, we show that with an increase in error rate of a few percentage points, the SNNs can achieve more than 2x reductions in operations compared to the original CNNs. This highlights the potential of SNNs in particular when deployed on power-efficient neuromorphic spiking neuron chips, for use in embedded applications.[1]

### 2.1.1 Introduction

Deep ANN architectures such as GoogLe-Net (Szegedy *et al.*, 2015) and VGG-16 (Simonyan *et al.*, 2014c) have successfully pushed the state-of-the-art classification error rates to new levels on challenging computer vision benchmarks like ImageNet (Russakovsky *et al.*, 2015). Inference in such very large networks, i. e. classification of an ImageNet frame, requires substantial computational and energy costs, thus limiting their use in mobile and embedded applications.

1 This section is an extended version of Rueckauer *et al.*, 2017.

Recent work has shown that the event-based mode of operation in SNNs is particularly attractive for reducing the latency and computational load of deep neural networks (Farabet *et al.*, 2012; Neil *et al.*, 2016; O'Connor *et al.*, 2013; Zambrano *et al.*, 2016). Deep SNNs can be queried for results already after the first output spike is produced, unlike ANNs where the result is available only after all layers have been completely processed (Diehl *et al.*, 2015b). SNNs are also naturally suited to process input from event-based sensors (Liu *et al.*, 2015; Posch *et al.*, 2014), but even in classical frame-based machine vision applications such as object recognition or detection, they have been shown to be accurate, fast, and efficient, in particular when implemented on neuromorphic hardware platforms (Esser *et al.*, 2016; Neil *et al.*, 2014; Stromatias *et al.*, 2015). SNNs could thus play an important role in supporting, or in some cases replacing deep ANNs in tasks where fast and efficient classification in real-time is crucial, such as detection of objects in larger and moving scenes, tracking tasks, or activity recognition (Hu *et al.*, 2016).

Multi-layered spiking networks have been implemented on digital commodity platforms such as FPGAs (Gokhale *et al.*, 2014; Neil *et al.*, 2014), but spiking networks with more than tens of thousands of neurons can be implemented on large-scale neuromorphic spiking platforms such as TrueNorth (Benjamin *et al.*, 2014; Merolla *et al.*, 2014) and SpiNNaker (Furber *et al.*, 2014). Recent demonstrations with TrueNorth (Esser *et al.*, 2016) show that CNNs of over a million neurons can be implemented on a set of chips with a power dissipation of only a few hundred mW. Given the recent successes of deep networks, it would be advantageous if spiking forms of deep ANN architectures such as VGG-16 can be implemented on these power-efficient platforms while still producing good error rates. This would allow the deployment of deep spiking networks in combination with an event-based sensor for real-world applications (Kiselev *et al.*, 2016; Orchard *et al.*, 2015b; Serrano-Gotarredona *et al.*, 2015).

In order to bridge the gap between Deep Learning continuous-valued networks and neuromorphic spiking networks, it is necessary to develop methods that yield deep SNNs with equivalent error rates as their continuous-valued counterparts. Successful approaches include direct training of SNNs using backpropagation (Lee *et al.*, 2016), the SNN classifier layers using stochastic gradient descent (Stromatias *et al.*, 2017), or modifying the transfer function of the ANNs during training so that the network parameters can be mapped better to the SNN (Esser *et al.*, 2015; Hunsberger *et al.*, 2016; O'Connor *et al.*, 2013). The largest architecture trained by (Hunsberger *et al.*, 2016) in this way is based on AlexNet (Krizhevsky *et al.*, 2012). While the results are promising, these novel methods have yet to mature to the state where training spiking architectures of the size of VGG-16 becomes possible, and the same state-of-the-art error rate as the equivalent ANN is achieved.

A more straightforward approach is to take the parameters of a pre-trained ANN and to map them to an equivalent-accurate SNN. Early studies on ANN-to-SNN conversion began with the work of (Perez-Carrasco *et al.*, 2013), where CNN units were translated into biologically inspired spiking units with leaks and refractory periods, aiming for processing inputs from event-based sensors. (Cao *et al.*, 2015) suggested a close link between the transfer function of a spiking neuron, i. e. the relation between input current and output firing frequency to the activation of

a ReLU, which is nowadays the standard model for the neurons in ANNs. They report good performance error rates on conventional computer vision benchmarks, converting a class of CNNs that was restricted to having zero bias and only average-pooling layers. Their method was improved by (Diehl *et al.*, 2015b), who achieved nearly loss-less conversion of ANNs for the MNIST (LeCun *et al.*, 1998) classification task by using a *weight normalization* scheme. This technique rescales the weights to avoid approximation errors in SNNs due to either excessive or too little firing of the neurons. (Hunsberger *et al.*, 2016) introduced a conversion method where noise injection during training improves the robustness to approximation errors of the SNN with more realistic biological neuron models. (Esser *et al.*, 2016) demonstrated an approach that optimized CNNs for the TrueNorth platform which has binary weights and restricted connectivity. (Zambrano *et al.*, 2016) have developed a conversion method using spiking neurons that adapt their firing threshold to reduce the number of spikes needed to encode information.

These approaches achieve very good results on MNIST, but the SNN results are below state-of-the-art ANN results when scaling up to networks that can solve CIFAR-10 (Krizhevsky, 2009). One reason is that in 2016, SNN implementations of many operators that are crucial for improved ANN error rate, such as max-pooling layers, softmax activation functions, and batch-normalization, were non-existent, and thus SNNs can only approximately match the inference of an ANN. As a consequence, none of the previously proposed conversion approaches are general enough for full automatic conversion of arbitrary pre-trained ANNs taken from a Deep-Learning model zoo available, for example, in Caffe[2].

In this work, we address some important shortcomings of existing ANN-to-SNN conversion methods. Through mathematical analysis of the approximation of the output firing rate of a spiking neuron to the equivalent analog activation value, we were able to derive a theoretical measure of the error introduced in the previous conversion process. On the basis of this novel theory, we propose modifications to the spiking neuron model that significantly improve the performance of deep SNNs. By developing spiking implementations of max-pooling layers, softmax activation, neuron biases, and batch normalization (Ioffe *et al.*, 2015), we extend the suite of CNNs that can be converted. In particular, we demonstrate for the first time that GoogLeNet Inception-V3 can be converted to an equivalent-accurate SNN. Further, we show that the conversion to spiking networks is synergistic with ANN network compression techniques such as parameter quantization and the use of low-precision activations.

To automate the process of transforming a pre-trained ANN into an SNN, we developed an SNN-conversion toolbox that is able to transform models written in Keras (Chollet, 2015), Lasagne and Caffe, and offers built-in simulation tools for evaluation of the spiking model. Alternatively, the converted SNN can be exported for use in spiking simulators like pyNN or Brian2. The documentation and source code is publicly available online[3].

The remainder of the paper is organized as follows: Section 2.1.2.1 outlines the conversion theory and Section 2.1.2.4 presents the methods for implementing the

2 https://github.com/BVLC/caffe/wiki/Model-Zoo

3 http://snntoolbox.readthedocs.io/

different features of a CNN. The work in these two sections is extended from earlier work in (Rueckauer *et al.*, 2016b). Section 2.1.3 presents the conversion results of networks tested on the MNIST, CIFAR-10, and ImageNet datasets.

### 2.1.2 Methods

#### 2.1.2.1 *Theory for Conversion of ANNs into SNNs*

The basic principle of converting ANNs into SNNs is that firing rates of spiking neurons should match the graded activations of analog neurons. (Cao *et al.*, 2015) first suggested a mechanism for converting ReLU activations, but a theoretical groundwork for this principle was lacking. Here we present an analytical explanation for the approximation, and on its basis we are able to derive a simple modification of the reset mechanism following a spike, which turns each SNN neuron into an unbiased approximator of the target function (Rueckauer *et al.*, 2016b).

We assume here a one-to-one correspondence between an ANN unit and a SNN neuron, even though it is also possible to represent each ANN unit by a population of spiking neurons. For a network with $L$ layers let $\mathbf{W}^l, l \in \{1, \dots, L\}$ denote the weight matrix connecting units in layer $l-1$ to layer $l$, with biases $\mathbf{b}^l$. The number of units in each layer is $M^l$. The ReLU activation of the continuous-valued neuron $i$ in layer $l$ is computed as

$$a_i^l := \max\left(0, \sum_{j=1}^{M^{l-1}} W_{ij}^l a_j^{l-1} + b_i^l\right), \tag{2.1}$$

starting with $\mathbf{a}^0 = \mathbf{x}$, where $\mathbf{x}$ is the input, normalized so that each $x_i \in [0,1]$.[4] Each SNN neuron has a membrane potential $V_i^l(t)$, which integrates its input current at every time step

$$z_i^l(t) := V_{\text{thr}}\left(\sum_{j=1}^{M^{l-1}} W_{ij}^l \Theta_{t,j}^{l-1} + b_i^l\right) \quad , \tag{2.2}$$

where $V_{\text{thr}}$ is the threshold and $\Theta_{t,i}^l$ is a step function indicating the occurrence of a spike at time $t$:

$$\Theta_{t,i}^l := \Theta(V_i^l(t-1) + z_i^l(t) - V_{\text{thr}}), \text{ with } \Theta(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{else.} \end{cases} \tag{2.3}$$

Every input pattern is presented for $T$ time steps, with time step size $\Delta t \in \mathbb{R}^+$. The highest firing rate supported by a time stepped simulator is given by the inverse time resolution $r_{\text{max}} := 1/\Delta t$. Input rates to the first layer are proportional to the constant pixel intensities or RGB image values. We can compute the firing rate of

4 This analysis focuses on applications with image data sets, which are generally transformed in this way. The argument could be extended to the case of zero-centered data by interpreting negative input to the first hidden layer of the SNN as coming from a class of inhibitory neurons, and inverting the sign of the charge deposited in the post-synaptic neuron.

each SNN neuron $i$ as $r_i^l(t) := N_i^l(t)/t$, where $N_i^l(t) := \sum_{t'=1}^{t} \Theta_{t',i}^l$ is the number of spikes generated.

The principle of the ANN-to-SNN conversion method as introduced in (Cao *et al.*, 2015; Diehl *et al.*, 2015b), postulates that the firing rate of a neuron $r_i^l$ correlates with its original ANN activation $a_i^l$ in (2.1). In the following, we introduce a membrane equation for the spiking neurons to formalize a concrete relationship $r_i^l(t) \propto a_i^l$.

**MEMBRANE EQUATION** The spiking neuron integrates inputs $z_i^l(t)$ until the membrane potential $V_i^l(t)$ exceeds a threshold $V_{\text{thr}} \in \mathbb{R}^+$ and a spike is generated. Once the spike is generated, the membrane potential is reset. We discuss next two types of reset: *reset to zero*, used e. g. in (Diehl *et al.*, 2015b), always sets the membrane potential back to a baseline, typically zero. *Reset by subtraction*, or "linear reset mode" in (Cassidy *et al.*, 2013; Diehl *et al.*, 2016a), subtracts the threshold $V_{\text{thr}}$ from the membrane potential at the time when it exceeds the threshold:

$$V_i^l(t) = \begin{cases} \left(V_i^l(t-1) + z_i^l(t)\right)\left(1 - \Theta_{t,i}^l\right) & \text{reset to zero} \quad (2.4a) \\ V_i^l(t-1) + z_i^l(t) \quad - V_{\text{thr}}\Theta_{t,i}^l & \text{reset by subtr.} \quad (2.4b) \end{cases}$$

From these membrane equations, we can derive slightly different approximation properties for the two reset mechanisms. In this section we analyze the first hidden layer and expand the argument in Section 2.1.2.1 to higher layers. We assume that the input currents $z_i^1 > 0$ remain constant over time, and justify this assumption in Section 2.1.2.4. The input to first-layer neurons (2.2) is then related to the ANN activations (2.1) via $z_i^1 = V_{\text{thr}} a_i^1$. In order to relate these ANN activations to the SNN spike rates, we merely have to average the membrane equations (2.4a) and (2.4b) over the simulation time. The detailed calculations are given in the Supplementary Material; the resulting rates are obtained as

$$r_i^1(t) = \begin{cases} a_i^1 r_{\max} \cdot \dfrac{V_{\text{thr}}}{V_{\text{thr}} + \epsilon_i^1} - \dfrac{V_i^1(t)}{V_{\text{thr}} + \epsilon_i^l} \cdot \dfrac{1}{t} & \text{reset to zero} \quad (2.5a) \\ a_i^1 r_{\max} \quad - \quad \dfrac{V_i^1(t)}{V_{\text{thr}}} \cdot \dfrac{1}{t} & \text{reset by subtr.} \quad (2.5b) \end{cases}$$

As expected, the spike rates are proportional to the ANN activations $a_i^1$, but reduced by an additive approximation error term, and in case of *reset to zero* an additional multiplicative error term. In the *reset to zero* case, with constant input, there is always a constant number of time steps $n_i^1$ between spikes of the same neuron $i$, and the threshold will always be exceeded by the same constant amount $\epsilon_i^1 = V_i^1(n_i^1) - V_{\text{thr}} = n_i^1 \cdot z_i^1 - V_{\text{thr}} \geq 0$. This residual charge $\epsilon_i^1$ is discarded at reset, which results in a reduced firing rate and thereby loss of information. For shallow networks and small datasets such as MNIST, this error seems to be a minor problem but we have found that an accumulation of approximation errors in deeper layers degrades the classification error rate. We also see from Eq. (2.5a) that a larger $V_{\text{thr}}$ and smaller inputs improve the approximation at the expense of longer integration times. Using the definition $(n_i^1 - 1)z_i^1 < V_{\text{thr}} \leq n_i^1 z_i^1$ for $n_i^1$ and $\epsilon_i^1 = n_i^1 z_i^1 - V_{\text{thr}}$ for

$\epsilon_i^1$, we find that the approximation error is limited from above by the magnitude of the input $z_i^1$. This insight further explains why the weight normalization scheme of (Diehl *et al.*, 2015b) improves performance in the reset-to-zero case: By guaranteeing that the ANN activations $a_i^1$ are too low to drive a neuron in the SNN above $V_{\text{thr}}$ within a single time step, we can keep $z_i^1 = V_{\text{thr}} a_i^1$ and thereby $\epsilon_i^l$ low. Another obvious way of improving the approximation is to reduce the simulation time step, but this comes at the cost of increased computational effort.

A simple switch to the *reset by subtraction* mechanism improves the approximation, and makes the conversion scheme suitable also for deeper networks. The excess charge $\epsilon$ is not discarded at reset and can be used for the next spike generation. Accordingly, the error term due to $\epsilon$ does not appear in Eq. (2.5b). Instead, the firing rate estimate in the first hidden layer converges to its target value $a_i^1 \cdot r_{\text{max}}$; the only approximation error due to the discrete sampling vanishes over time. We validate by simulations in Section 2.1.3.1 that this mechanism indeed leads to more accurate approximations of the underlying ANN than the methods proposed in (Cao *et al.*, 2015; Diehl *et al.*, 2015b), in particular for larger networks.

**FIRING RATES IN HIGHER LAYERS** The previous results were based on the assumption that the neuron receives a constant input $z$ over the simulation time. When neurons in the hidden layers are spiking, this condition only holds for the first hidden layer and for inputs in the form of analog currents instead of irregular spike trains. In the *reset-by-subtraction* case, we can derive analytically how the approximation error propagates through the deeper layers of the network. For this, we insert the expression for SNN input $z_i^l$ from Eq. 2.2 into the membrane equation Eq. (2.4b) for $l > 1$, average $V_i^l(t)$ over the simulation time, and solve for the firing rate $r_i^l(t)$. This yields

$$r_i^l(t) = \sum_{j=1}^{M^{l-1}} W_{ij}^l r_j^{l-1}(t) + r_{\text{max}} b_i^l - \frac{V_i^l(t)}{V_{\text{thr}}} \cdot \frac{1}{t}. \tag{2.6}$$

This equation states that the firing rate of a neuron in layer $l$ is given by the weighted sum of the firing rates of the previous layer, minus the time-decaying approximation error described in Eq. (2.5b). This relationship implies that each layer computes a weighted sum of the approximation errors of earlier layers, and adds its own approximation error. The recursive expression Eq. (2.6) can be solved iteratively by inserting the expression for the previous layer rates, starting with the known rates of the first layer Eq. (2.5b):

$$r_i^l = a_i^l r_{\text{max}} - \Delta V_{i_l}^l - \sum_{i_{l-1}=1}^{M^{l-1}} W_{i_l i_{l-1}}^l \Delta V_{i_{l-1}}^{l-1} - \cdots \\ - \sum_{i_{l-1}=1}^{M^{l-1}} W_{i_l i_{l-1}}^l \cdots \sum_{i_1=1}^{M^1} W_{i_2 i_1}^2 \Delta V_{i_1}^1 \tag{2.7}$$

with $\Delta V_i^l := V_i^l(t)/(V_{\text{thr}} \cdot t)$. Thus, a neuron $i$ in layer $l$ receives an input spike train with a slightly lower spike rate, reduced according to the quantization error $\Delta V$ of previous layer neurons. These errors accumulate for higher layers, which explains

why it takes longer to achieve high correlations of ANN activations, and why SNN firing rates deteriorate in higher layers.

#### 2.1.2.2 *Parameter Normalization*

One source of approximation errors is that in time-stepped simulations of SNNs, the neurons are restricted to a firing rate range of $[0, r_{\max}]$, whereas ANNs typically do not have such constraints. Weight normalization is introduced by (Diehl *et al.*, 2015b) as a means to avoid approximation errors due to too low or too high firing. This work showed significant improvement of the performance of converted SNNs by using a *data-based* weight normalization mechanism. We extend this method to the case of neurons with biases and suggest a method that makes the normalization process more robust to outliers.

**NORMALIZATION WITH BIASES.** The *data-based* weight normalization mechanism is based on the linearity of the ReLU used for ANNs. It can simply be extended to biases by linearly rescaling all weights and biases such that the ANN activation $a$ (as computed in Eq. (2.1)) is smaller than 1 for all training examples. In order to preserve the information encoded within a layer, the parameters of a layer need to be scaled jointly. Denoting the maximum ReLU activation in layer $l$ as $\lambda^l = \max[\mathbf{a}^l]$, then weights $\mathbf{W}^l$ and biases $\mathbf{b}^l$ are normalized to $\mathbf{W}^l \rightarrow \mathbf{W}^l \frac{\lambda^{l-1}}{\lambda^l}$ and $\mathbf{b}^l \rightarrow \mathbf{b}^l / \lambda^l$.

**ROBUST NORMALIZATION.** Although weight normalization avoids firing rate saturation in SNNs, it might result in very low firing rates, thereby increasing the latency until information reaches the higher layers. We refer to the algorithm described in the previous paragraph as "max-norm", because the normalization factor $\lambda^l$ was set to the maximum ANN activation within a layer, where the activations are computed using a large subset of the training data. This is a very conservative approach, which ensures that the SNN firing rates will most likely not exceed the maximum firing rate. The drawback is that this procedure is prone to be influenced by singular outlier samples that lead to very high activations, while for the majority of the remaining samples, the firing rates will remain considerably below the maximum rate.

Such outliers are not uncommon, as shown in Fig. 2.1, which plots the log-scale distribution of all non-zero activations in the first convolution layer for 16,666 CIFAR-10 samples. The maximum observed activation is more than three times higher than the 99.9th percentile. Figure 2.2 shows the distribution of the highest activations across the 16,666 samples for all ANN units in the same layer, revealing a large variance across the dataset, and a peak that is far away from the absolute maximum. This distribution explains why normalizing by the maximum can result in a potentially poor classification performance of the SNN. For the vast majority of input samples, even the maximum activation of units within a layer will lie far below the chosen normalization scale leading to insufficient firing within the layer to drive higher layers and subsequently worse classification results.

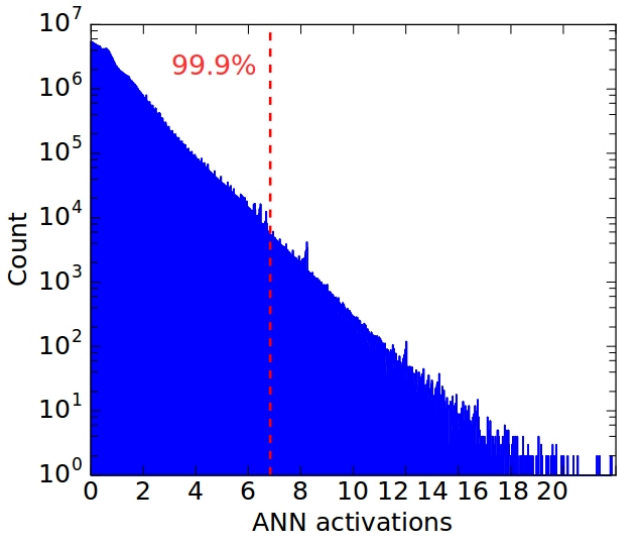


**Figure 2.1:** Distribution of all non-zero activations in the first convolution layer of a CNN, for 16666 CIFAR-10 samples, and plotted in log-scale. The dashed line in both plots indicates the 99.9th percentile of all ReLU activations across the dataset, corresponding to a normalization scale $\lambda = 6.83$. This is more than three times less than the overall maximum of $\lambda_{max} = 23.16$.

We propose a more robust alternative where we set $\lambda^l$ to the $p$-th percentile of the total activity distribution of layer $l$.[5] This choice discards extreme outliers, and increases SNN firing rates for a larger fraction of samples. The potential drawback is that a small percentage of neurons will saturate, so choosing the normalization scale involves a trade-off between saturation and insufficient firing. In the following, we refer to the percentile $p$ as the "normalization scale", and note that the "max-norm" method is recovered as the special case $p = 100$. Typical values for $p$ that perform well are in the range $[99.0, 99.999]$. In general, saturation of a small fraction of neurons leads to a lower degradation of the network classification error rate compared to the case of having spike rates that are too low. This method can be combined with Batch-Normalization (BN) used during ANN training (Ioffe *et al.*, 2015), which normalizes the activations in each layer and therefore produces fewer extreme outliers.

#### 2.1.2.3 *Spikes With Payload*

Unlike in biology, an n-bit spike in silicon may carry additional information beyond the binary event, for instance an estimate of the firing rate, or the presynaptic membrane potential. In section 2.1.2.1 we found that the sampling of a continuous spike-rate with discrete events introduces an approximation error that is propagated through the network and corrupts the performance of the SNN. In the following

5 This distribution is obtained by computing the ANN activations on a large fraction of the training set. From this, the scaling factor can be determined and applied to the layer parameters. This has to be done only once for a given network; during inference the parameters do not change.

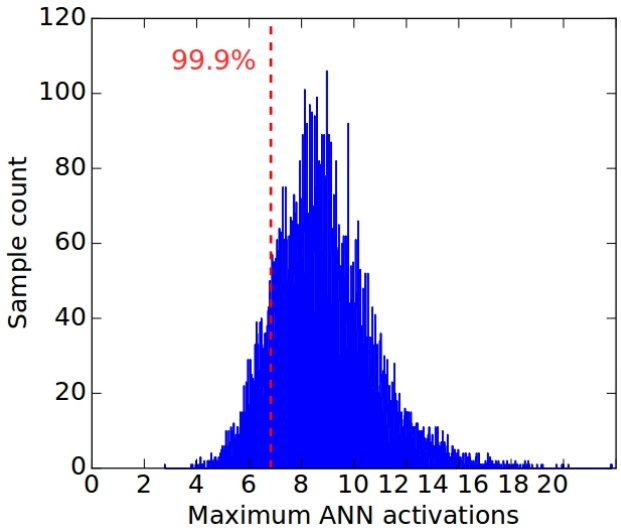


**Figure 2.2:** Distribution of maximum ReLU activations for the same 16666 CIFAR-10 samples. For most samples their maximum activation is far from $\lambda_{max}$.

we determine an expression for the payload that could be sent with the spike in order to alleviate this degradation.[6]

**ANSATZ** As ansatz we add a term to the membrane equation (2.4b) that describes the payload mechanism: Each time a presynaptic neuron $j$ of layer $l-1$ fires a spike, a payload $\alpha_{ij}^{l-1}$ is added[7] to the membrane potential of the postsynaptic neuron $i$ of layer $l$.

$$V_i^l(t) = V_i^l(t-1) + V_{\text{thr}} \left( \sum_{j=1}^{M^{l-1}} W_{ij}^l \Theta_{t,j}^{l-1} + b_i^l \right) - V_{\text{thr}} \Theta_{t,i}^l + \sum_{j=1}^{M^{l-1}} W_{ij}^l \Theta_{t,j}^{l-1} \alpha_{ij}^{l-1}(t), \quad (2.8)$$

**PAYLOAD IN FIRST LAYER** We first consider the payload terms from layer 1 to layer 2, and generalize the result to layer $l$ later. The first hidden layer makes an error $(V_i^1(t) - V_i^1(0))/(V_{\text{thr}} \cdot t)$ when estimating the target firing rate $a_i^1/dt = z_i^1/(V_{\text{thr}} dt)$ at time $t$ (see Eq. (2.5b)). As before, we can average the membrane equation over time to get the firing rate:

$$r_i^2(t) = \sum_{j=1}^{M^1} W_{ij}^2 r_j^1(t) + \frac{b_i^2}{dt} - \frac{V_i^2(t) - V_i^2(0)}{V_{\text{thr}}} \cdot \frac{1}{t} + \frac{1}{V_{\text{thr}}} \frac{1}{t} \sum_{t'=1}^{t} \sum_{j=1}^{M^1} W_{ij}^2 \Theta_{t',j}^1 \alpha_{ij}^1(t'). \quad (2.9)$$

6 This work about spikes with payload was done after completing the publication (Rueckauer *et al.*, 2017) on which the rest of this chapter is based.

7 To simplify notation later on, we choose to weight this payload by the synaptic strength $W_{ij}^l$, but this factor could have been absorbed in the definition of $\alpha$ and does not lessen the generality of this ansatz. Also, the dependence of the payload term $\alpha_{ij}$ on the post-synaptic neuron $i$ is unrealistic from a practical perspective but keeps the ansatz general. It will fall away later.

Inserting the expression for the rates of layer 1 (2.5b):

$$r_i^2(t) = \frac{1}{dt}\left(\sum_{j=1}^{M^1} W_{ij}^2 a_j^1 + b_i^2\right) - \frac{V_i^2(t) - V_i^2(0)}{V_{\text{thr}}} \cdot \frac{1}{t} \tag{2.10}$$
$$- \sum_{j=1}^{M^1} W_{ij}^2 \frac{V_j^1(t) - V_j^1(0)}{V_{\text{thr}}} \cdot \frac{1}{t} + \frac{1}{V_{\text{thr}}} \frac{1}{t} \sum_{t'=1}^{t} \sum_{j=1}^{M^1} W_{ij}^2 \Theta_{t',j}^1 \alpha_{ij}^1(t').$$

The first term equals precisely the target rate $a_i^2$. The second term corresponds to the error that neuron $i$ in layer 2 makes when estimating this target rate using discrete spikes. The third term is the error of the previous layer neurons, weighted by the connection strength. With the payload $\alpha$ in the last term we try to eliminate these approximation errors. We need to realize however that, by causality, the payload $\alpha_{ij}^1$ from the first hidden layer cannot possibly make up for the error introduced by the succeeding hidden layer (second term). Thus we focus on the last two terms when deriving the expression for the payload.

**PAYLOAD OF A SINGLE SPIKE** We approach the solution iteratively, defining $t = t_1$ as the first time where at least one of the $M^1$ neurons in layer 1 spikes. The step function $\Theta_{t,j}^1$ is zero for all $t < t_1$, and equals one only for the set of neurons $J^1(t_1) = \{j \mid j \text{ fires at time } t_1\} \subseteq \{1, \dots, M^1\}$. The set of payload terms to accompany the spikes is given by the following minimization:

$$\boldsymbol{\alpha}^1(t_1) = \{\alpha_{ij}^1(t_1) \mid j \in J^1(t_1)\} = \underset{\boldsymbol{\alpha}^1(t_1)}{\operatorname{argmin}} |q_1|, \tag{2.11}$$

$$q_1 := \sum_{j \in J^1(t_1)} W_{ij}^2 \alpha_{ij}^1(t_1) - \sum_{j'=1}^{M^1} W_{ij'}^2 \left(V_{j'}^1(t_1) - V_{j'}^1(0)\right) \tag{2.12}$$

The minimization of $q_1$ requires that the combined payload of the subset of spiking neurons cancel out the combined membrane potentials of all presynaptic neurons. The two sums in $q$ can be redistributed by introducing the complementary set $\bar{J}^1(t_1)$ of layer 1 neurons that do not spike at time $t_1$:

$$q_1 = \sum_{j \in J^1(t_1)} W_{ij}^2 \left(\alpha_{ij}^1(t_1) - \left(V_j^1(t_1) - V_j^1(0)\right)\right) - \sum_{j' \in \bar{J}^1(t_1)} W_{ij'}^2 \left(V_{j'}^1(t_1) - V_{j'}^1(0)\right) \tag{2.13}$$

By construction, a payload is attributed only to spiking neurons, and is oblivious to the state of the other neurons in the same layer, in particular their membrane potential, whether or not they spike, or what payload they might be sending simultaneously. Thus, the payload $\boldsymbol{\alpha}^1$ has no means to account for the approximation error contained in non-spiking neurons of the same layer, represented by the second sum above. Thus, the best correction we can possibly achieve with a payload is to minimize the first sum. Each individual summand becomes zero simply by setting the payload of neuron $j$ to the current state of the membrane potential:

$$\alpha_j^1(t_1) = V_j^1(t_1) - V_j^1(0) \tag{2.14}$$

This result is reasonable because the payload then corresponds to the residual of the membrane potential after spike, i. e. the sampling error of estimating the continuous firing rate. Also, there is no dependence on the target neuron $i$, as expected.

Note that $\alpha_j^1(t_1) \geq 0 \; \forall j \in J^1(t_1)$ because they belong to neurons whose membrane potential was just above threshold. Thus the payload of presynaptic neuron $j$ added to the membrane potential of postsynaptic neuron $i$ compensates for the reduced firing rate of $j$. However, as explained above, this correction is not perfect as it can not account for the error in non-spiking neurons. Payload correction will be better the more active a layer is.

**PAYLOAD OF LATER SPIKES** So far we have determined the payload of the first spike. How does the payload change for a spike at a later time $t_N$? For this we again consider the last two terms of Eq. (2.10), which constitute our cost function to be minimized:

$$q_{1:N} = \sum_{t'=1}^{t} \sum_{j=1}^{M^1} W_{ij}^2 \Theta_{t',j}^1 \alpha_{ij}^1(t') - \sum_{j'=1}^{M^1} W_{ij'}^2 \left( V_{j'}^1(t_N) - V_{j'}^1(0) \right) \tag{2.15}$$

The time integration can be rewritten as a sum over the spike times $\{t_1, ..., t_N\}$:

$$q_{1:N} = \sum_{t=t_1}^{t_N} \sum_{j \in J^1(t)} W_{ij}^2 \alpha_{ij}^1(t) - \sum_{j'=1}^{M^1} W_{ij'}^2 \left( V_{j'}^1(t_N) - V_{j'}^1(0) \right) \tag{2.16}$$

By separating the payload term of current spike time $t_N$, we can make use of the previous result for a single spike (c. f. (2.12)):

$$\begin{aligned} q_{1:N} &= \sum_{t=t_1}^{t_{N-1}} \sum_{j \in J^1(t)} W_{ij}^2 \alpha_{ij}^1(t) + \sum_{j \in J^1(t_N)} W_{ij}^2 \alpha_{ij}^1(t_N) - \sum_{j'=1}^{M^1} W_{ij'}^2 \left( V_{j'}^1(t_N) - V_{j'}^1(0) \right) \\ &= \sum_{t=t_1}^{t_{N-1}} \sum_{j \in J^1(t)} W_{ij}^2 \alpha_{ij}^1(t) + q_N \end{aligned} \tag{2.17}$$

Now the expression to be minimized by the payload contains the term for a single time of spike that we encountered before, plus the accumulated payload from all previous spikes. In order to minimize this expression, the current payload $\alpha_j^1(t_N)$ should be reduced by the previous corrections:

$$\alpha_j^1(t_N) = V_j^1(t_N) - V_j^1(0) - \sum_{t=t_1}^{t_{N-1}} \alpha_j^1(t). \tag{2.18}$$

This corrective term does not cancel out the entire first term in Eq. (2.17), but only the summands related to neuron $j$. This makes sense from a practical perspective: A payload can easily be implemented to remember and use the sum of its own previous values, but taking into account the history of its neighboring neurons would require additional structure, like lateral connections. Thus, the payload of neuron $j \in J^1(t_N)$ at time $t_N$ can only account for the previous corrections of this same neuron $j$, i. e. the first term in equation (2.17) will only be reduced if $j$ is also an element of some set of previously spiking neurons $J^1(t < t_N)$. This observation is in line with our event-based construction of the payload mechanism: Corrections are updated only when there is a spike.

The recursive formula (2.18) for the payload at an arbitrary time of spike can be simplified into the explicit form $\alpha_j^l(t_N) = V_j^l(t_N) - V_j^l(t_{N-1})$ using the initial condition (2.14).

**PAYLOAD IN HIGHER LAYERS** The expressions for the payload in the first hidden layer need not change for higher layers, because the approximation error of layer 1 has been canceled out as much as possible, so that the payload of succeeding layers only has to deal with the errors introduced by their respective layer. Thus, the general expression for the payload using a reset-by-subtraction is simply given by the voltage increase that led to a spike at time $t_N$:

$$\alpha_j^l(t_N) = V_j^l(t_N) - V_j^l(t_{N-1}). \tag{2.19}$$

This result makes sense intuitively: From the perspective of the postsynaptic neuron $i$, a tiny voltage increase in presynaptic neuron $j$ will look no different than a large voltage increase - both result in a single spike at $t_N$. If the voltage step was large, any excess charge above threshold will be preserved thanks to the subtractive reset, but it may take several time steps before the excess voltage contributes to another spike. With a payload proportional to the final voltage step, the postsynaptic neuron is informed immediately about a stronger signal.

We expect this mechanism to reduce the discretization error that accompanies rate coding with binary spikes. With reduced errors, the simulation duration is likely to be reduced as well. Unfortunately, our experimental results are limited to MNIST, where the margin of improvement is too small to provide a definitive answer of the usefulness of a spike payload in practice. A more rigorous evaluation is left for future work.

#### 2.1.2.4 *Spiking Implementations of ANN Operators*

In this section we introduce new methods that improve the classification error rate of deep SNNs (Rueckauer *et al.*, 2016b). These methods either allow the conversion of a wider ranger of ANNs, or reduce the approximation errors in the SNN.

**CONVERTING BIASES** Biases are standard in ANNs, but were explicitly excluded by previous conversion methods for SNNs. In a spiking network, a bias can simply be implemented with a constant input current of equal sign as the bias. Alternatively, one could present the bias with an external spike input of constant rate proportional to the ANN bias, as proposed in (Neftci *et al.*, 2014). The theory in Section 2.1.2.1 can be applied to the case of neurons with biases, and the following Section 2.1.2.2 shows how parameter normalization can be applied to biases as well.

**CONVERSION OF BATCH-NORMALIZATION LAYERS** BN reduces internal covariate shift in ANNs and thereby speeds up the training process. BN introduces additional layers where affine transformations of inputs are performed in order to achieve zero-mean and unit variance. An input $x$ is transformed into $\mathrm{BN}[x] = \frac{\gamma}{\sigma}(x - \mu) + \beta$, where mean $\mu$, variance $\sigma$, and the two learned parameters $\beta$ and $\gamma$ are all obtained during training as described in (Ioffe *et al.*, 2015). After training, these transformations can be integrated into the weight vectors, thereby preserving the effect of BN, but eliminating the need to compute the normalization repeatedly for each sample during inference. Specifically, we set $\tilde{W}_{ij}^l = \frac{\gamma_i^l}{\sigma_i^l} W_{ij}^l$ and $\tilde{b}_i^l = \frac{\gamma_i^l}{\sigma_i^l}\left(b_i^l - \mu_i^l\right) + \beta_i^l$. This makes it simple to convert BN layers into SNNs, because after transforming the

weights of the preceding layer, no additional conversion for BN layers is necessary. Empirically we found loss-less conversion when the BN parameters are integrated into other weights. The advantage lies purely in obtaining better performing ANNs if BN is used during training.

**ANALOG INPUT TO FIRST LAYER** Because event-based benchmark datasets are rare (Hu *et al.*, 2016; Rueckauer *et al.*, 2016a), conventional frame-based image datasets such as MNIST (LeCun *et al.*, 1998) and CIFAR (Krizhevsky, 2009) have been used to evaluate the classification error rate of the converted SNN. Previous methods (Cao *et al.*, 2015; Diehl *et al.*, 2015b) usually transform the analog input activations, e. g. gray levels or RGB values, into Poisson firing rates. But this transformation introduces variability into the firing of the network and impairs its performance.

Here, we interpret the analog input activations as constant currents. Following Eq. (2.2), the input to the neurons in the first hidden layer is obtained by multiplying the corresponding kernels with the analog input image $\mathbf{x}$:

$$z_i^1 := V_{\text{thr}} \left( \sum_{j=1}^{M^0} W_{ij}^1 x_j + b_i^1 \right) \quad . \tag{2.20}$$

This results in one constant charge value $z_i^l$ per neuron $i$, which is added to the membrane potential at every time step. The spiking output then begins with the first hidden layer. Empirically we found this to be particularly effective in the low-activation regime of ANN units, where usually undersampling in spiking neurons poses a challenge for successful conversion.

**SPIKING SOFTMAX** Softmax is commonly used on the outputs of a deep ANN, because it results in normalized and strictly positive class likelihoods. Previous approaches for ANN-to-SNN conversion did not convert softmax layers, but simply predicted the output class corresponding to the neuron that spiked most during the presentation of the stimulus. However, this approach fails when all neurons in the final layer receive negative inputs, and thus never spike.

Here we implement two versions of a spiking softmax layer. The first is based on the mechanism proposed in (Nessler *et al.*, 2009), where output spikes are triggered by an external Poisson generator with fixed firing rate. The spiking neurons do not fire on their own but simply accumulate their inputs. When the external generator determines that a spike should be produced, a softmax competition according to the accumulated membrane potentials is performed. The second variant of our spiking softmax function is similar, but does not rely on an external clock. To determine if a neuron should spike, we compute the softmax on the membrane potentials, and use the resulting values in range of $[0, 1]$ as rate parameters in a Poisson process for each neuron. In both variants, the final classification result over the course of stimulus presentation is then given by the index of the neuron with the highest firing rate, as before. We prefer the second variant because it does not depend on an additional hyperparameter and is not stochastic. A third variant has been suggested by one of the reviewers: Since the softmax is applied at the last layer of the network, one could simply infer the classification output from the softmax computed on the membrane potentials, without another spike generation

mechanism. This simplification could speed up inference time and possibly improve the accuracy by reducing stochasticity. This method is appealing where one does not insist upon a purely spiking network.

**SPIKING MAX-POOLING LAYERS** Most successful ANNs use max-pooling to spatially down-sample feature maps. However, this has not been used in SNNs because computing maxima with spiking neurons is non-trivial. Instead, simple average pooling used in (Cao *et al.*, 2015; Diehl *et al.*, 2015b), results in weaker ANNs being trained before conversion. Lateral inhibition, as suggested in (Cao *et al.*, 2015), does not fulfill the job properly, because it only selects the winner, but not the actual maximum firing rate. Another suggestion is to use a temporal Winner-Take-All based on time-to-first-spike encoding, in which the first neuron to fire is considered the maximally firing one (Masquelier *et al.*, 2007; Orchard *et al.*, 2015c). Here we propose a simple mechanism for spiking max-pooling, in which output units contain gating functions that only let spikes from the maximally firing neuron pass, while discarding spikes from other neurons. The gating function is controlled by computing estimates of the pre-synaptic firing rates, e. g. by computing an online or exponentially weighted average of these rates. In practice we found several methods to work well, but demonstrate only results using a finite impulse response filter to control the gating function.

#### 2.1.2.5 *Counting Operations*

To obtain the number of operations in the networks during classification, we define as fan-in $f_{\text{in}}$ the number of incoming connections to a neuron, and similarly fan-out $f_{\text{out}}$ as the number of outgoing projections to neurons in the subsequent layer. To give some examples: In a convolutional layer, the fan-in is given by the size of the 2-dimensional convolution kernel multiplied by the number of channels in the previous layer. In a fully-connected layer, the fan-in simply equals the number of neurons in the preceding layer. The fan-out of a neuron in a convolutional layer $l$ that is followed by another convolution layer $l+1$ generally depends on the stride of layer $l+1$. If the stride is 1, the fan-out is simply given by the size of the 2-dimensional convolution kernel of layer $l+1$, multiplied by the number of channels in layer $l+1$. Note that the fan-out may be reduced in corners and along edges of the feature map depending on how much padding is applied.

In case of the ANN, the total number of floating-point operations for classification of one frame is given by

$$\sum_{l=1}^{L}(2f_{\text{in},l}+1)n_l \quad \text{Ops/frame}, \tag{2.21}$$

with $n_l$ the number of neurons in layer $l$. The factor 2 comes from the fact that each fan-in operation consist of a multiplication and addition. With $+1$, we count the operations needed to add the bias. The pooling operation is not considered here.

In the case of an SNN, only additions are needed when the neuron states are updated. We adopt the notation from (Merolla *et al.*, 2014) and report the *Synaptic Operations*, i. e. the updates in the neurons of a layer caused by a spike in the

previous layer.[8] The total number of synaptic operations in the SNN across the simulation duration $T$ is

$$\sum_{t=1}^{T}\left[\sum_{l=1}^{L} f_{\text{out},l}\, s_l(t)\right] \quad \text{Ops/frame}, \tag{2.22}$$

where $s_l(t)$ denotes the number of spikes fired in layer $l$ at time $t$.

In the ANN, the number of operations needed to classify one image, consisting of the cost of a full forward-pass, is a constant. In the SNN, the image is presented to the network for a certain simulation duration, and the network outputs a classification guess at every time step. By measuring both the classification error rate and the operation count at each step during simulation, we are able to display how the classification error rate of the SNN gradually decreases with increasing number of operations (cf Fig. 2.7).

The two different modes of operation - single forward pass in the ANN vs. continuous simulation in the SNN - have significant implications when aiming for an efficient hardware implementation. One well known fact is that additions required in SNNs are cheaper than multiply accumulates needed in ANNs. For instance, our simulations in a Global Foundry 28 nm process show that the cost of performing a 32-bit floating-point addition is about 14X lower than that of a MAC operation and the corresponding chip area is reduced by 21X. It has also been shown that memory transfer outweighs the energy cost of computations by two orders of magnitude (Horowitz, 2014). In the ANN, reading weight kernels and neuron states from memory, and writing states back to memory is only done once during the forward pass of one sample. In contrast, memory access in the SNN is less predictable and has to be repeated for individual neurons in proportion to their spike rates. If the number of operations needed by the SNN to achieve a similar classification error as that of the ANN is lower, then equivalently the SNN would also have a reduction in the number of memory accesses. The direct implementation of SNNs on dedicated spiking hardware platforms like SpiNNaker or TrueNorth is left to future work, and will be necessary for estimating the real energy cost in comparison to the cost of implementing the original ANNs on custom ANN hardware accelerators like Eyeriss (Chen *et al.*, 2016).

### 2.1.3 Results

There are two ways of improving the classification error rate of an SNN obtained via conversion: 1) training a better ANN before conversion, and 2) improving the conversion by eliminating approximation errors of the SNN. We proposed several techniques for these two approaches in Sec. 2.1.2; in Sections 2.1.3.1 and 2.1.3.2 we evaluate their effect using the CIFAR-10 data set. Sec. 2.1.3.3 extends the SNN conversion methods to the ImageNet data set. In Sec. 2.1.3.4 we show that SNNs feature an accuracy-vs-operations trade-off that allow tuning the performance of a network to a given computational budget.

8 This *synaptic* operation count does not include updates of the state variables due to a bias or dynamics of the post-synaptic potential (which is instantaneous in our case). We validated in our simulations that the operations caused by the bias are about two orders of magnitude fewer in number than synaptic operations, in addition to being less costly in terms of memory fetches).

The networks were implemented in Keras (Chollet, 2015). Some of the CIFAR-10 results were previously reported in (Rueckauer *et al.*, 2016b).

#### 2.1.3.1 *Contribution of Improved ANN Architectures*

The methods introduced in Section 2.1.2 allow conversion of CNNs that use biases, softmax, batch-normalization, and max-pooling layers, which all improve the classification error rate of the ANN. The performance of a converted network was quantified on the CIFAR-10 benchmark (Krizhevsky, 2009), using a CNN with 4 convolution layers (32 3x3 - 32 3x3 - 64 3x3 - 64 3x3), ReLU activations, batch-normalization, 2x2 max-pooling layers after the 2nd and 4th convolutions, followed by 2 fully connected layers (512 and 10 neurons respectively) and a softmax output. This ANN achieved 12.14% error rate (Table 2.1). Constraining the biases to zero increased the error rate to 12.27%. Replacing max-pooling by average-pooling further decreased the performance to 12.31%. Eliminating the softmax and using only ReLUs in the output led to a big drop to 30.56%. With our new methods we can therefore start the conversion already with much better ANNs than was previously possible.

#### 2.1.3.2 *Contribution of Improved SNN Conversion Methods*

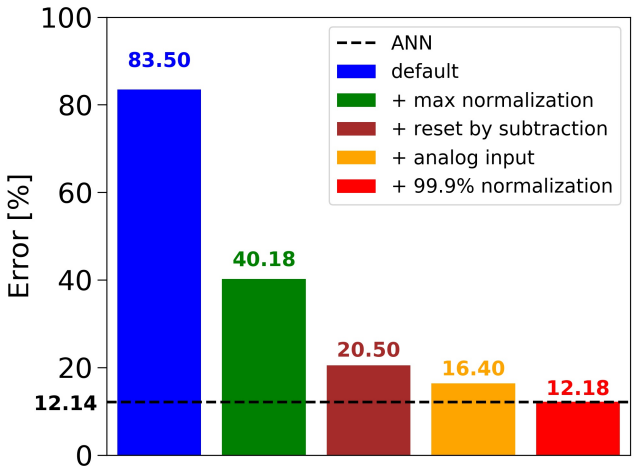


**Figure 2.3:** Influence of novel mechanisms for ANN-to-SNN conversion on the SNN error rate for CIFAR-10.

Figure 2.3 shows that in the case of CIFAR-10, the conversion of the best ANN into an SNN using the default approach (i. e. no normalization, Poisson spike train input, reset-to-zero) fails, yielding an error rate of 83.50%, barely above chance level. Adding the data-based weight normalization (Diehl *et al.*, 2015b) (green bar) lowers the error rate to 40.18%, but this is still a big drop from the ANN result of 12.14% (dashed black line). Changing to the *reset-by-subtraction* mechanism from Section 2.1.2.1 leads to another 20% improvement (brown bar), and switching to analog inputs to the first hidden layer instead of Poisson spike trains results in an error rate of 16.40% (orange bar). Finally, using the 99.9th percentile of activations

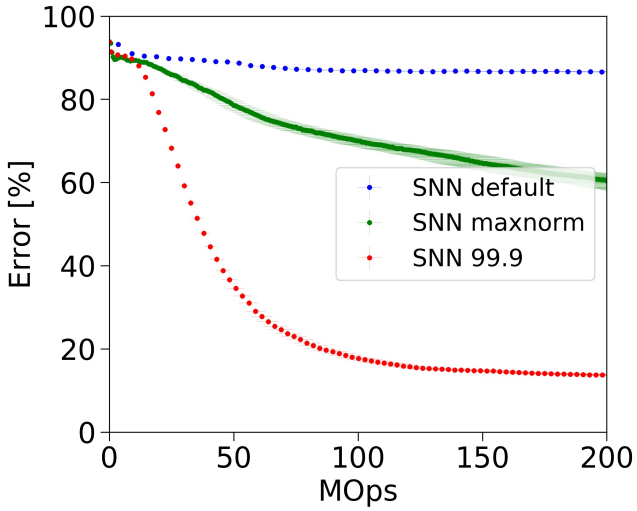


**Figure 2.4:** Accuracy-latency trade-off. Robust parameter normalization (red) enables our spiking network to correctly classify CIFAR-10 samples much faster than using our previous max-normalization (green). Not normalizing leads to classification at chance level (blue).

for robust weight normalization yields 12.18% error rate, which is on par with the ANN performance and gives our best result for CIFAR-10. We therefore conclude that our proposed mechanisms for ANN training and ANN-to-SNN conversion contribute positively to the success of the method. The conversion into a SNN is nearly loss-less, and the results are very competitive for classification benchmarks using SNNs (Table 2.1). These results were confirmed also on MNIST, where a 7-layer network with max-pooling achieved an error rate of 0.56%, thereby improving previous state-of-the-art results for SNNs reported by (Diehl *et al.*, 2015b) and (Zambrano *et al.*, 2016).

SNNs are known to exhibit a so-called accuracy-latency trade-off (Diehl *et al.*, 2015b; Neil *et al.*, 2016), which means that the error rate drops the longer the network is simulated, i. e. the more operations we invest. The latency in which the final error rate is achieved, is dependent on the type of parameter normalization as illustrated by the three curves in Figure 2.4. Parameter normalization is necessary to improve upon chance-level classification (blue, no normalization). However, our previous max-norm method (green) converges very slowly to the ANN error rate because the weight scale is overly reduced and spike-activity is low. With a robust normalization using the 99.9th percentile of the activity distribution, the weights are larger and convergence is much faster. Empirically, the best results were obtained with normalization factors in the range between the 99th and 99.9th percentile of activations, which allows the network to converge quickly to error rates similar to those of the underlying ANN.

This accuracy-latency trade-off is very prominent in case of the classic LeNet architecture on MNIST (Fig. 2.5). While the ANN achieves an error rate of 1.04 % using a fixed amount of 2.35 MOps per frame, the spiking model reaches within 1 percentage point of the ANN using 2x less operations (2.07 % error rate at 1.07

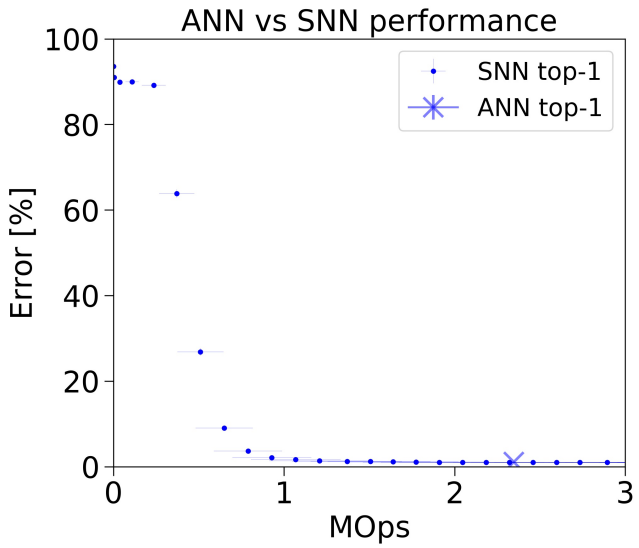


**Figure 2.5:** Classification error rate vs number of operations for the LeNet ANN and SNN implementation on the MNIST dataset.

MOps/frame). At 1.47 MOps, the SNN error rate is 1.13 %. The SNN then continues to improve until it reaches 1.07 % error rate at the end of the simulation.

#### 2.1.3.3 *ImageNet*

VGG (Simonyan *et al.*, 2014c) and GoogLeNet (Szegedy *et al.*, 2015) are two deep network architectures that won first places in the localization and classification competitions of the ImageNet ILSVRC-2014 respectively. By introducing *inception modules* and *bottlenecks*, GoogLeNet requires 12X fewer parameters and significantly less computes than VGG-16, even though the total layer count is much higher. Since their initial introduction in 2014, both architectures have been improved. The third version of GoogLeNet which was released in 2015 as Inception-V3 (Szegedy *et al.*, 2016), improved on the ImageNet results to state-of-the art 5.6% top-5 error rate, and uses 2.5X more computes than the original GoogLeNet. This was in part done by further reducing the kernel size and dimensions inside the network, applying regularization via batch-normalized auxiliary classifiers, and label smoothing.

**TRANSIENT DYNAMICS AND VOLTAGE CLAMP** While the conversion pipeline outlined in Section 2.1.2 can deliver converted SNNs that produced equivalent error rates as the original ANNs on the MNIST and CIFAR-10 data sets, the error rate of the converted Inception-V3 was initially far from the error rate of the ANN. One main reason is that neurons undergo a transient phase at the beginning of the simulation because a few neurons have large biases or large input weights. During the first few time steps, the membrane potential of each neuron needs to accumulate input spikes before it can produce any output. The firing rates of neurons in the first layer need several time steps to converge to a steady rate, and this convergence time is increased in higher layers that receive transiently varying input. The convergence time is decreased in neurons that integrate high-frequency input, but increased in

neurons integrating spikes at low frequency.[9] Another factor contributing to a large transient response are 1x1 convolution layers. In these layers, the synaptic input to a single neuron consists only of a single column through the channel-dimension of the previous layer, so that the neuron's bias or a single strongly deviating synaptic input may determine the output dynamics. With larger kernels, more spikes are gathered that can outweigh the influence of e. g. a large bias.[10]

In order to overcome the negative effects of transients in neuron dynamics, we tried a number of possible solutions, including the initializations of the neuron states, different reset mechanisms, and bias relaxation schemes. The most successful approach we found was to clamp the membrane potential to zero for the first $N$ time-steps, where $N$ increases linearly with the layer depth $l$: $N(l) = d \cdot l$. The slope $d$ represents the temporal delay between lifting the clamp from consecutive layers. The longer the delay $d$, the more time is given to a previous layer to converge to steady-state before the next layer starts integrating its output.

This simple modification of the SNN state variables removes the transient response completely (Fig. S1), because by the time the clamp is lifted from postsynaptic neurons, the presynaptic neurons have settled at their steady-state firing-rate. We found a clamping delay of $d = 10$ in Inception-V3 to be sufficient. Clamping the membrane potential in VGG-16 did not have a notable impact on the error rate. Each input image was presented to the converted VGG-16 spiking network for 400 time steps, and to the converted Inception-V3 for 550 time steps. The average firing rate of neurons is 0.016[11] in VGG-16, and 0.053 Hz in Inception-V3. Note that the clamping delay does not necessarily result in an increased execution time of the SNN: The reduced approximation errors in Inception-V3 allow the accuracy to converge within a shorter simulation duration. There is likely a sweet spot for choosing the clamping delay such as to minimize the runtime while maximizing accuracy. A more systematic evaluation of this trade-off is left for future work.

We expect that the transient of the network could be reduced by training the network with constraints on the biases or the $\beta$ parameter of the batch-normalization layers. Table 2.1 summarizes the error rates achieved by our SNNs using the methods presented above, and compares them to previous work by other groups.

#### 2.1.3.4 *Combination with Low-Precision Models*

The neurons in our spiking network emit events at a rate proportional to the activation of the corresponding unit in the ANN. Target activations with reduced precision can be approximated more quickly and accurately with a small num-

9 An ANN neuron responds precisely the same whether (A) receiving input from a neuron with activation 0.1 and connecting weight 0.8, or (B) activation 0.8 and weight 0.1. In contrast, the rate of an SNN neuron will take longer to converge in case (A) than in (B). This phenomenon forms the basis of the accuracy-latency trade-off mentioned above: One would like to keep firing rates as low as possible to reduce the operational cost of the network, but has to sacrifice approximation accuracy for it.

10 Even though the parameters in each layer were normalized such that the input to each neuron is below threshold, this does not guarantee that all biases are sub-threshold: their effect could be reduced by inhibitory input spikes. While such inhibitory synaptic input is still missing at the onset of the simulation, the output dynamics of a neuron will be dominated by a large bias.

11 As our neuron model does not contain any time constant, this unit should be read as "spikes per simulation time step" and is not related to spikes per wall-clock time.

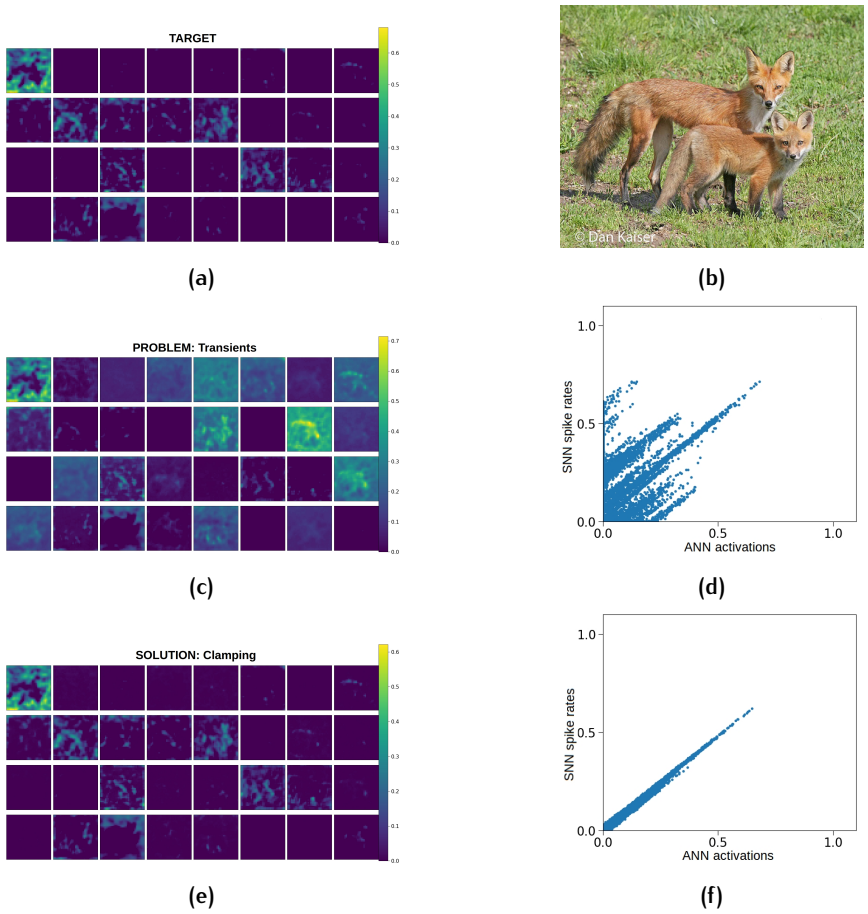


**Figure 2.6:** **(a)** Inception-V3 activations in layer 14, consisting of 32 1x1 convolutions. This heat map represents the target rates which the SNN approximates. **(b)** The input image. **(c)** SNN spike-rates of the same layer, when the membrane potential of the SNN is not clamped at the start. Clearly, some feature maps are more active than their corresponding ANN feature maps, other maps are less active. The consistent offset across a feature map points to the bias as the plausible cause for this offset. **(d)** Correlation between ANN activations from (a) and SNN spike rates from (c) reveal the offsets of entire feature maps. **(e)** SNN spike rates of the same layer, with membrane potentials clamped for 10 time steps between consecutive layers. The resulting spike rates are much closer to the ANN target activations. **(f)** ANN-SNN correlation is close to perfect.

ber of spike events. For instance, if the activations are quantized into values of $\{0, 0.1, 0.2, ..., 0.9, 1.0\}$, the spiking neuron can perfectly represent each value within at most 10 time steps. On the other hand, to approximate a floating-point precision number using 16 bit precision, the neuron in the worst case would have to be active for $2^{16} = 65536$ time steps.

To demonstrate the potential benefit of using low-precision activations when transforming a given model into a spiking network, we apply the methods from Sec. 2.1.2.4 to BinaryNet (Courbariaux *et al.*, 2016), a CNN where both weights and activations are constrained to either $\{0, +1\}$, or $\{-1, +1\}$. To obtain the binarized ANNs with these two sets of activations, we train BinaryNet using the publicly available source code (Courbariaux *et al.*, 2016) on two different activation functions: First with a Heaviside activation function, and second, with a signed activation function. The two binarized models are then converted into spiking networks. Instead of interpreting the negative activations of BinaryNet "sign" as negative firing rates, we invert the sign of the spikes emitted by neurons with a negative activation. To achieve this, we add a second threshold at $-1$, where neurons can emit spikes of size $-1$ if the threshold is reached from above.

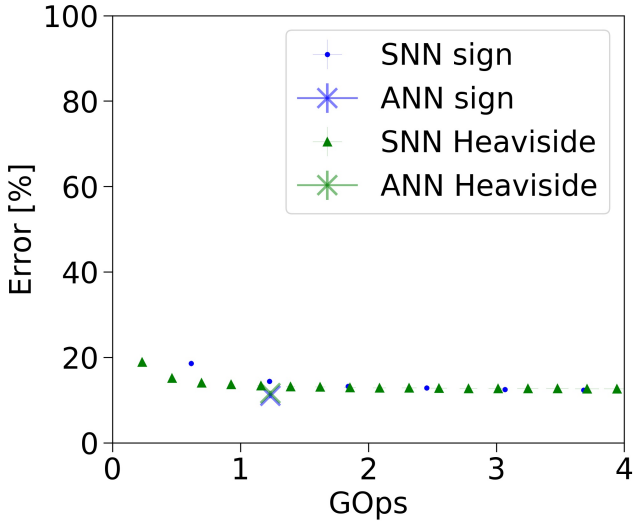


**Figure 2.7:** Classification error rate vs number of operations for the BinaryNet ANN and SNN implementation on the complete CIFAR-10 dataset.

By virtue of the quantized activations, these two SNNs are able to approximate the ANN activations with very few operations (see Fig. 2.7). The BinaryNet SNNs already show an error rate which is close to the ANN target error rates early in the simulation, in fact as soon as the first output spikes are produced. In contrast, in full-precision models (cf. Figs. 2.4 and 2.5), the classification error rate starts at chance level and drops over the course of the simulation, as more operations are invested.

The lowest error rate for our converted spiking CIFAR-10 models is achieved using BinaryConnect (Courbariaux *et al.*, 2015). This network is trained using full-precision weights in combination with binarized weights. Either set of weights can be used during inference. We test the resulting model with both the binarized weights and

the full-precision copy kept during training (c. f. Table 2.1). These results illustrate how spiking networks benefit from and at the same time complement the strengths of low-precision models.

### 2.1.4 Discussion

This work presents two new developments. The first is a novel theory that describes the approximation of an SNN firing rates to its equivalent ANN activations. The second is the techniques to convert almost arbitrary continuous-valued CNNs into spiking equivalents. By implementing SNN-compatible versions of common ANN features such as max pooling, softmax, batch normalization, biases and Inception modules, we allow a larger class of CNNs including VGG-16 and GoogLeNet Inception-V3 to be converted into SNNs. Table 2.1 shows that our SNN results compare favorably to previous SNN results on all tested data sets: (Cao *et al.*, 2015) achieved 22.57% error rate on CIFAR-10, albeit with a smaller network and after cropping images to 24x24. With a similarly small network and cropped images, (Hunsberger *et al.*, 2016) achieve 16.46% error rate. Better SNN error rates to date have only been reported by (Esser *et al.*, 2016), where an error rate of 12.50% was reported for a very large network optimized for 8 TrueNorth chips, and making use of ternary weights and multiple 1x1 network-in-network layers. A smaller network fitting on a single chip is reported to achieve 17.50%. In our own experiments with similar low-precision training schemes for SNNs, we converted the BinaryConnect model by (Courbariaux *et al.*, 2016) to 8.65% error rate on CIFAR-10, which is by far the best SNN result reported to date.

In addition to the improved SNN results on MNIST and CIFAR-10, this work presents for the first time, a spiking network implementation of VGG-16 and Inception-V3 models, utilizing simple non-leaky integrate-and-fire neurons. The top-5 error rates of the SNNs during inference lie close to the original ANNs. Future investigations will be carried out to identify additional conversion methods that will allow the VGG-16 SNN to reach the error rate of the ANN. For instance, we expect a reduction in the observed initial transients of higher up layers within large networks, by training the networks with constraints on the biases.

With BinaryNet (an 8-layer CNN with binary weights and activations tested on CIFAR-10) (Courbariaux *et al.*, 2016), we demonstrated that low-precision models are well suited for conversion to spiking networks. While the original network requires a fixed amount of 1.23 GOps to classify a single frame with average error rate of 11.57%, the SNN can be queried for a classification result at a variable number of operations. For instance, the average error rate of the SNN is 15.13% at 0.46 GOps (2.7x reduction), and improves further when investing more operations. This reduction in operation count is due to the fact that, first, activation values at lower precision can more easily be approximated by discrete spikes, and second, zero activations are natively skipped in the activity-driven operation of spiking networks. In light of this, our work builds upon and complements the recent advances in low-precision models and network compression.

The converted networks highlight a remarkable feature of spiking networks: While ANNs require a fixed amount of computations to achieve a classification

**Table 2.1:** Classification error rate on MNIST, CIFAR-10 and ImageNet for our converted spiking models, compared to the original ANNs, and compared to spiking networks from other groups. The reported error rate is top-1, with top-5 in brackets for ImageNet. The values in bold highlight the best SNN result for a particular data set.

| Data set [architecture] | ANN err. | SNN err. | neur. | synap. |
|---|---|---|---|---|
| MNIST [ours] | 0.56 | **0.56** | 8 k | 1.2 M |
| MNIST [(Zambrano *et al.*, 2016)] | 0.86 | 0.86 | 27 k | 6.6 M |
| CIFAR-10 [ours, BinaryNet sign] | 11.03 | 11.75 | 0.5 M | 164 M |
| CIFAR-10 [ours, BinaryNet Heav] | 11.58 | 12.55 | 0.5 M | 164 M |
| CIFAR-10 [ours, BinaryConnect, binarized at infer.] | 16.81 | 16.65 | 0.5 M | 164 M |
| CIFAR-10 [ours, BinaryConnect, full prec. at infer.] | 8.09 | **9.15** | 0.5 M | 164 M |
| CIFAR-10 [ours] | 11.13 | 11.18 | 0.1 M | 23 M |
| CIFAR-10 [(Esser *et al.*, 2016)], 8 chips | NA | 12.50 | 8 M | NA |
| CIFAR-10 [(Esser *et al.*, 2016)], single chip | NA | 17.50 | 1 M | NA |
| CIFAR-10 [(Hunsberger *et al.*, 2016)]* | 14.03 | 16.46 | 50 k | NA |
| CIFAR-10 [(Cao *et al.*, 2015)]** | 20.88 | 22.57 | 35 k | 7.4 M |
| ImageNet [ours, VGG-16]† | 36.11 (15.14) | 50.39 (18.37) | 15 M | 3.5 B |
| ImageNet [ours, Inception-V3]†† | 23.88 (7.01) | **25.40** (**7.96**) | 11.7 M | 0.5 B |
| ImageNet [(Hunsberger *et al.*, 2016)]‡ | NA | 48.20 (23.80) | 0.5 M | NA |

*Cropped to 24x24. **Cropped to 24x24. † On a subset of 2570 samples, using single-scale images of size 224x224. †† On a subset of 1382 samples, using single-scale images of size 299x299. ‡On a subset of 3072 samples.

result, the final error rate in a spiking network drops off rapidly during inference when an increasing number of operations is used to classify a sample. The network classification error rate can be tailored to the number of operations that are available during inference, allowing for accurate classification at low latency and on hardware systems with limited computational resources. In some cases, the number of operations needed for correct classification can be reduced significantly compared to the original ANN. We found a savings in computes of 2x for smaller full-precision networks (e. g. LeNet has 8 k neurons and 1.2 M connections), and larger low-precision models (e. g. BinaryNet has 0.5 M neurons and 164 M connections). These savings did not scale up to the very large networks such as VGG-16 and Inception-V3 with more than 11 M neurons and over 500 M connections. One reason is that each additional layer in the SNN introduces another stage where high-precision activations need to be approximated by discrete spikes. We show in Eq. 2.5b that this error vanishes over time. But since higher layers are driven by inputs that contain approximation errors from lower layers (cf. Eq. 2.6), networks of increasing depth need to be simulated longer for an accurate approximation. We are currently investigating spike encoding schemes that make more efficient use of temporal structure than the present rate-based encoding. (Mostafa *et al.*, 2017) present such an approach where the precise spike time is used to train a network to classify MNIST digits with a single spike per neuron. Such a sparse temporal code clearly reduces the cost of repeated weight fetches which dominates in rate-encoded SNNs.

Finally, this conversion framework allows the deployment of state-of-the-art pre-trained high-performing ANN models onto energy-efficient real-time neuromorphic spiking hardware such as TrueNorth (Benjamin *et al.*, 2014; Merolla *et al.*, 2014; Pedroni *et al.*, 2016).

## 2.2 LATENCY CODE

The activations of an ANN are usually treated as representing an analog firing rate. When mapping the ANN onto an equivalent SNN, this rate-based conversion can lead to undesired increases in computation cost and memory access, if firing rates are high. This work presents an efficient temporal encoding scheme, where the analog activation of a neuron in the ANN is treated as the instantaneous firing rate given by the TTFS in the converted SNN. By making use of temporal information carried by a single spike, we show a new spiking network model that uses $7-10\times$ fewer operations than the original rate-based analog model on the MNIST handwritten dataset, with an accuracy loss of $< 1\%$.[12]

### 2.2.1 Introduction

DNNs achieve state-of-the-art in numerous machine learning applications, for instance object detection and classification, scene parsing, and video captioning (LeCun *et al.*, 2015; Schmidhuber, 2014). Highly accurate classification during inference comes at a cost: Typical ANNs require floating-point MAC operations to compute the activation values of all the neurons in the network. Graphics Processing Units (GPUs) process these operations efficiently in parallel, but their power consumption can prohibit their use in embedded applications. Considerable effort is being devoted to the development of hardware accelerators as well as algorithmic improvements which enable the efficient execution of deep neural networks on these hardware platforms. Notable directions include reducing the precision of weights and / or activations (Hubara *et al.*, 2016), skipping computations when activation values are zero (Aimar *et al.*, 2019), and minimizing data movement (Chen *et al.*, 2016).

Unmatched in power efficiency, the biological brain employs all-or-none pulses (spikes) to transmit information. Motivated by this paradigm, artificial SNNs are being developed to solve similar tasks as ANNs, but making use of cheaper "addition" operations instead of MACs, and leveraging sparsity in neuron activations, as neurons will only be active if driven by a strictly positive input. In tasks with a continuous stream of input data, SNNs combined with event-based sensors (Lichtsteiner *et al.*, 2008) allow for data-driven computation, making SNNs ideal for always-on, low-power applications. Neuromorphic hardware (Furber *et al.*, 2014; Merolla *et al.*, 2014) optimizes routing of spikes across the network and implements power-efficient computation.

Training deep SNNs directly can be difficult. Recent spike-based learning algorithms demonstrated promising results on the MNIST dataset (Lee *et al.*, 2016; Mostafa, 2018), and even natural image datasets (Kheradpisheh *et al.*, 2018), but the scalability of the learning algorithm to larger architectures and more challenging data sets has yet to be shown. Mapping a pre-trained ANN to an SNN of similar architecture has been applied successfully to natural image datasets like CIFAR-10 (Rueckauer *et al.*, 2016b), and ImageNet (Hunsberger *et al.*, 2016; Rueckauer *et al.*, 2017). The encoding scheme underlying this SNN conversion approach is rate-based:

12 This section is based on Rueckauer *et al.*, 2018.

The SNN neurons generate a sequence of discrete spikes, which, when averaged over the simulation duration, approximates the analog firing rate of the corresponding ANN neuron. This rate-based encoding becomes more accurate as the simulation duration of the SNNs is increased and more spikes are generated. The computational cost of the SNN also scales with the average firing rate of its neurons: Each spike entails fetching the weight values of the synaptic connections to neurons in the following layer, and updating the state variables (e. g. , membrane potential) of those post-synaptic neurons. In the ANN, *all* neurons are updated *once*, using MAC operations; in the SNN, an *active subset* of neurons is updated *repeatedly*, using ADD operations. Because the energy cost of memory transfer can exceed the energy cost of computations by two orders of magnitude (Horowitz, 2014), the SNN may lose a potential performance advantage over the ANN as the rates increase.

The energy cost disparity of memory and computation motivates the search for a spike code that is more efficient than the rate-based code. Coupled with evidence of temporal codes in the brain (VanRullen *et al.*, 2001), we propose in this paper an ANN-to-SNN conversion mechanism, where the analog activation values of the ANN neurons are represented by the inverse Time-To-First-Spike (TTFS) in the SNN neurons (Thorpe *et al.*, 2001). Thus, neurons in the SNN spike at most once during inference of one sample, combining the spatial (feature map) sparsity of SNNs with the temporal sparsity of ANNs.

Earlier work in this direction includes the HFirst (Orchard *et al.*, 2015c) network which uses spike timing in object recognition: The max-pool operation is implemented by performing a temporal winner-take-all among neurons in a pooling region. The authors in Zambrano *et al.*, 2016 developed a method to convert ANNs to SNNs based on rate-coding, but with adapting thresholds that reduce the firing rates significantly compared to other rate-based methods. Several groups have trained SNNs directly, either in an unsupervised (Diehl *et al.*, 2015a; Kheradpisheh *et al.*, 2018) or supervised manner (Mostafa, 2018; Zhao *et al.*, 2015). Closely related to our coding scheme, Mostafa *et al.*, 2017 proposed an algorithm to train an SNN using an analytic expression for the Time-To-First-Spike (TTFS), which can be optimized via back-propagation. Where applicable, we compare their results with ours in Sec. 2.2.3.

The three variants of our proposed TTFS encoding are described in Sec. 2.2.2. The evaluation of these methods on the MNIST dataset is presented in Sec. 2.2.3, and the results discussed in Sec. 2.2.4.

### 2.2.2 Methods

#### 2.2.2.1 *TTFS Baseline Model ("TTFS base")*

In general, the dynamics of the membrane potential $u_i(t)$ of a neuron $i$ can be framed within the Spike Response Model (Gerstner *et al.*, 2014):

$$u_i(t) = \sum_{t_i^{(f)} \in \mathcal{F}_i} \eta_i(t - t_i^{(f)}) + \sum_{j \in \Gamma_i} \sum_{t_j^{(f)} \in \mathcal{F}_j} w_{ij} \epsilon_{ij}(t - t_j^{(f)}) + I_{\text{ext}}(t), \tag{2.23}$$

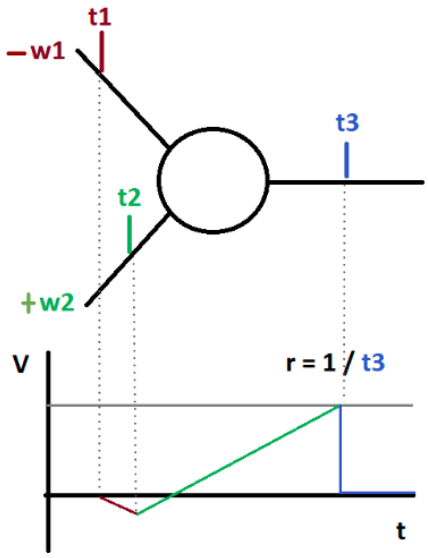


**Figure 2.8:** Spike generation mechanism with Time-To-First-Spike (TTFS) coding.

where $\eta_i$ models the shape of an action potential (including a possible refractory period), the scalar values $w_{ij}$ are the synaptic weights, the kernel $\epsilon_{ij}$ describes the Post-Synaptic Potential (PSP) caused by an input spike from neuron $j$, and the external input current $I_{\text{ext}}(t)$ represents the bias. The set $\mathcal{F}_i = \{t_i^{(f)} \mid 1 \leq f \leq n\} = \{t \mid u_i(t) = \theta\}$ contains the output spike times, and $\Gamma_i$ denotes the set of pre-synaptic neurons.

In this work, we require only the first spike of each neuron and can prevent additional spikes e. g. , by making the refractory period very long. Equation (2.23) for the membrane potential up to the first spike of neuron $i$ then simplifies to

$$u_i(t) = \sum_{j \in \Gamma_i} w_{ij} \epsilon_{ij}(t - t_j^{(0)}) + b_i t. \tag{2.24}$$

We choose a simple, non-leaky, piecewise-linear form for the PSP kernel $\epsilon_{ij}(t - t_j^{(0)}) = \left[t - t_j^{(0)}\right] \mathcal{H}(t - t_j^{(0)})$, where $\mathcal{H}(x)$ is the Heaviside step function.[13] The Heaviside function can be removed by introducing $\Gamma_i^< := \{j \mid t_j^{(0)} < t_i^{(0)}\}$ as the set of "causal neurons", i. e. , pre-synaptic neurons whose spikes arrive at neuron $i$ before its output spike is generated. Then

$$u_i(t) = \sum_{j \in \Gamma_i^<} w_{ij} \left[t - t_j^{(0)}\right] + b_i t. \tag{2.25}$$

13 For more biological realism, an alpha function could be used as PSP kernel. We decided against it to avoid the additional hyperparameters required (two time constants plus a factor for the kernel amplitude). Also, the linear kernel allows for an analytic treatment, and does not require a look-up table when implemented on hardware.

In a simulation with time step $dt$, this explicit equation translates to a rate of increase

$$\frac{u_i(t+dt) - u_i(t)}{dt} = \sum_{j \in \Gamma_i^<} w_{ij} + b_i =: \mu_i. \tag{2.26}$$

This mechanism is illustrated in Figure 2.8. In order to determine the first instance in time when neuron $i$ spikes, we set the membrane potential equal to the threshold, $u_i(t_i^{(0)}) = \theta$, and solve for $t_i^{(0)}$:

$$t_i^{(0)} = \frac{1}{\mu_i} \left( \theta + \sum_{j \in \Gamma_i^<} w_{ij} t_j^{(0)} \right) \tag{2.27}$$

The corresponding instantaneous firing rate $r_i$ equals $1/t_i^{(0)}$. Successful ANN-to-SNN conversion implies that the SNN spike rate $r_i$ is approximately equal to the corresponding activation $a_i$ of the original ANN.[14] Then the output spike time in the SNN is inversely proportional to the ANN activation: $t_i^{(0)} = 1/a_i$.

The update mechanism (2.25) constitutes the base model of the present work, labeled "TTFS base" in the remainder of the paper. A potential problem with the TTFS expression (2.27) derived from the base model can be seen by considering two neurons $A$, $B$ which drive neuron $C$ with connection strengths $w_A = 1$ and $w_B = -2$. If the two input neurons are activated with $a_A = 2$ and $a_B = 1$ respectively, the net effect on neuron $C$ in the ANN would cancel out. In the SNN however, neuron $A$ will fire twice as fast as $B$, potentially driving $C$ above threshold before the inhibitory input arrives. Raising the threshold to delay generation of output spikes in the post-synaptic neuron is not a viable solution because it will equally delay the arrival of input spikes.

#### 2.2.2.2 *TTFS with Dynamic Threshold ("TTFS dyn thresh")*

In order to remedy the situation where neurons fire their first spike prematurely, i. e. , before having received all input spikes, we replaced the constant threshold by a threshold that dynamically adapts to the observed input. In particular, the threshold of each neuron is increased by an amount equal to the input that is still missing. When an input spike arrives, the threshold is reduced by the amount equal to the corresponding synaptic strength. Thus, the likelihood of producing an output spike is inversely proportional to the information that would be lost if the spike were generated prematurely.

To determine the missing input, the pre-synaptic neuron $j$ signals to its target neuron $i$ whether the sign of its present PSP $x_j(t)$ is positive. If the net PSP of a neuron is excitatory (it expects to fire a spike in the future), then the post-synaptic neuron $i$ updates its threshold according to the following rule:

$$\theta_i(t) = \theta_i(\infty) + \sum_{j \in \Gamma_i} |w_{ij}| \mathcal{H}(x_j(t)). \tag{2.28}$$

14 We assume the ANN uses the common rectifying linear unit activation function $\mathrm{ReLU}(x) = \max(0, x)$.

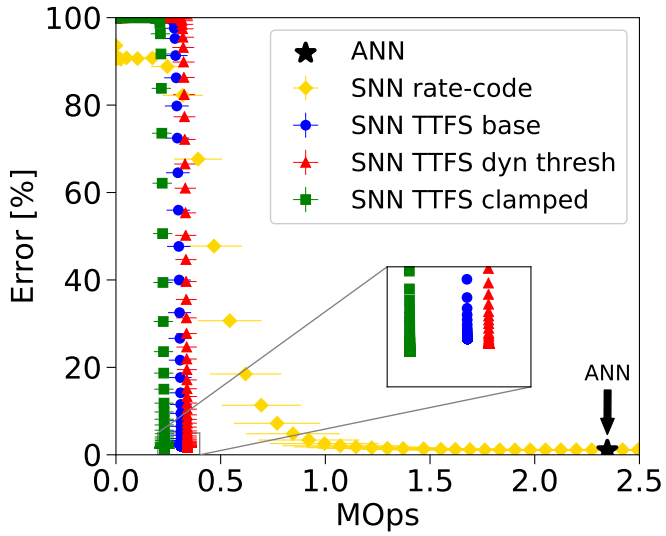


**Figure 2.9:** Classification error versus operation count of the original LeNet-5 ANN and the converted SNNs tested on MNIST. Error bars are removed from inset for clarity.

In essence, each neuron fires two types of spikes: The first type signals whether to expect a spike in the future, and the second type is the actual output spike. We label this method "TTFS dyn thresh" in the remainder of the paper.

#### 2.2.2.3 *TTFS with Clamped ReLU ("TTFS clamped")*

Because the spike count in "TTFS dyn thresh" is at least twice as high as in "TTFS base", we propose a second approach to mitigate the effect of long-latency spikes. The original ANN is refined with a modified ReLU activation function, where the activation values below some threshold $\beta > 0$ are clamped to zero:

$$\mathrm{ReLU}_{\mathrm{clamp}}(x) = \begin{cases} 0 & \text{if } x \leq \beta \\ x & \text{else.} \end{cases} \tag{2.29}$$

This way, the network learns to perform well without relying on low activations, and neurons in the converted SNN do not have to wait for long-latency spikes. A neat side effect is that sparsity in feature maps is increased, further reducing the spike count in the SNN. An obvious limitation of this method is the need to retrain with a constraint in the form of the clamped ReLU (2.29). This refinement is trivial in case of small problems like MNIST, and may even show additional benefits accompanying regularizations, but can become cumbersome for larger networks and datasets. This method is labeled "TTFS clamped" in the remainder of the paper.

### 2.2.3 Results

We tested the three versions of the Time-To-First-Spike (TTFS) approach ("TTFS base", "TTFS dyn thresh", "TTFS clamped") on the MNIST handwritten digit recognition data set and using the classic Lenet-5 (LeCun *et al.*, 1998) model. The dataset consists of

**Table 2.2:** Comparison with other SNNs on MNIST.

| | err [%] | | | |
|---|---|---|---|---|
| Method | ANN | SNN | # spikes | # neurons |
| Rate-based conversion | 1.40 | 1.52 | $10^4$ | 3194 |
| (Diehl *et al.*, 2015b) | 0.90 | 0.88 | $10^5$ | 5194 |
| Rate-based conversion | 1.16 | 1.16 | 200 | 3194 |
| (Zambrano *et al.*, 2016) | 0.86 | 0.86 | 100 | 5194 |
| STDP, TTFS (Kheradpisheh *et al.*, 2018) | NA | 1.60 | 600 | 9144 |
| STDP (Diehl *et al.*, 2015a) | NA | 5.0 | 17 | 6400 |
| Supervised TTFS learning (Mostafa *et al.*, 2017) | NA | 3.02 | 135 | 1384 |
| TTFS fully-conn. [this work] | 1.50 | 1.65 | 100 | 1394 |
| TTFS convol. [this work] | 1.04 | 1.43 | 1000 | 7614 |

70000 28x28 gray-scale images, of which 10000 were used for testing, the rest for training. The Lenet-5 model contains three convolution layers, two max-pooling layers and two fully-connected layers, amounting to a total of 7614 neurons and 1.2 million synapses. The network conversion and simulations were performed using the "SNN toolbox"[15]. The top-1 error of our LeNet-5 ANN is 1.04%; the classification of a single sample in one forward pass requires 2.35 million operations[16]. Inference in the SNN is not done in a single forward pass where all kernel operations and neuron updates are performed once, but instead consists of simulating the SNN until one of the 10 output neurons has spiked. Then the simulation is stopped. At each time step, we measure both the classification error and the cumulative number of spikes in the network. From the spike count we infer the total number of operations in the SNN: Each spike causes a number of synaptic operations equal to the number of post-synaptic neurons connected to it.

The curves of the mean classification error versus the number of operations for all 3 methods are shown in Fig. 2.9. We also display the curve obtained when converting the LeNet-5 ANN using a rate-code as in Rueckauer *et al.*, 2017. In all four SNNs, classification error is high initially while spikes propagate through the network and the membrane potential of output neurons is still below threshold. Once an output spike is produced, the classification error drops. This drop is remarkably steep in the TTFS encoded networks, indicating that such a network needs approximately the same number of operations to classify each sample. The error-curve of the rate-coded SNN drops more slowly over time because neurons get to fire multiple times, gradually improving the network output.

All three versions of the TTFS encoding scheme produce very similar results: Their error rates are all within 1% of that from the original ANN, while the number of operations is reduced by 7-10X. The "TTFS base" baseline version reduces operational

15 http://snntoolbox.readthedocs.io

16 MACs are counted twice, for multiplication and addition.

**Table 2.3:** LeNet-5 performance on MNIST.

| Model | MOps | Err [%] |
|---|---|---|
| ANN | 2.35 | 1.04 |
| SNN rate code (44 sim. steps) | 3.03 | 1.07 |
| SNN rate code (18 sim. steps) | 1.07 | 2.07 |
| SNN TTFS base | 0.31 | 2.00 |
| SNN TTFS dyn thresh | 0.34 | 1.80 |
| SNN TTFS clamped | 0.23 | 1.47 |

cost by a factor of 7, but the classification error rate is almost double due to missing long-latency spikes as described in Sec. 2.2.2. The advantage of this method is that we do not have to retrain the network just as for the "TTFS clamped" version, and we do not have to compute the threshold updates as in "TTFS dyn thresh". As explained in Sec. 2.2.2.2, threshold updates require transmission of a flag, indicating whether the post-synaptic neuron should expect a spike from the pre-synaptic neuron. The spike count of "TTFS dyn thresh" shown in Fig. 2.9 (not counting threshold updates) is higher than "TTFS base" because output spike generation is delayed to take into account long-latency spikes. When including threshold updates, the operation count of "TTFS dyn thresh" is 3X larger than the base-line, but still 2X lower than the ANN. The "TTFS clamped" approach achieves the lowest error rate at the lowest operational cost during inference, because the clamped ReLU increases sparsity in activations. This favorable number of operations does not take into account the one-time computational overhead when refining the ANN.

In order to compare our temporal coding method with the one presented in Mostafa *et al.*, 2017, we trained an ANN using the same architecture as in Mostafa *et al.*, 2017, namely a fully connected network with 784 input neurons, 600 hidden neurons, and 10 output neurons. The error rate of this ANN on MNIST is 1.50% and the operational cost is 953 kOps. Using the "TTFS base" method, the converted SNN achieves 1.65% error rate at 0.59 kOps. This computational cost is roughly equivalent to a network average of 100 spikes per sample. The method proposed in Mostafa *et al.*, 2017 does not convert an existing ANN, but trains the SNN directly with back-propagation, using an expression for the Time-To-First-Spike (TTFS) similar to Eq. (2.27). The gray-scale images were binarized and the bit-precisions of the network parameters and activations were reduced so as to facilitate the implementation on an FPGA. The cost function contains a penalty term that encourages neurons to fire early. Their resulting SNN achieves a 3.02% error rate, at a reported average activity of 135 spikes per sample.

### 2.2.4 Discussion

This work presents methods to convert analog neural networks into spiking neural networks using a temporal coding scheme that significantly reduces the spike redundancy present in previous rate-based conversion methods. In our proposed

implementation, the activation value of neurons in the ANN is encoded as Time-To-First-Spike (TTFS) in the converted SNN. Thus, every neuron fires at most once, if receiving a positive net input, but does not spike if receiving zero or negative input. This encoding is sparse in both time and feature space, making the model suitable for low-power embedded applications. The baseline TTFS method does not require modification of the network architecture or model parameters, and achieves a reduction in operation cost of 7X using LeNet-5 on MNIST. The classification error rate however increases by 1 percentage point. This increase can be mitigated by two proposed variants of the baseline method. The first variant consists of the refinement of the original ANN before conversion, by using a modified ReLU activation function. The second variant employs dynamic thresholds in the SNN to take into account the outputs of low-activated neurons. Both variants produce error rates close to the ANN error rate. The cost associated with this improvement is the need for retraining before inference, or for threshold updates during inference.

The SNNs obtained with the proposed TTFS code compare favorably against previous SNN results on MNIST (Tab. 2.2), obtained either by conversion or direct training of the SNN. The rate-based conversion of Diehl *et al.*, 2015b is superior in terms of error rate but requires significantly more spikes. The rate-based conversion by Zambrano *et al.*, 2016 is likewise more accurate, and features low spike rates due to its threshold adaptation. However, the spike rate alone does not account for the hidden computations due to threshold updates and processing of the real-valued spikes. Methods for training SNNs directly score in terms of low spike rates, but either require a higher number of neurons or converge to a higher error rate. We wish to emphasize the limited value of comparing conversion and training methods: Converting a pretrained ANN gives the SNN a "head-start" in classification performance; training the SNN directly has the advantage that the model can be taught to cope with artifacts that would occur during conversion, for instance the long-latency spikes discussed in Sec. 2.2.2.

SNNs potentially run more efficiently than standard ANNs for two reasons: (1) SNNs use additions instead of MACs, which consume about 14X less energy and occupy 21X less area[17]. (2) Zeros in feature maps are automatically skipped in the forward-pass of the SNN. On the other hand, SNN operation incurs additional memory traffic due to repeated updating of neuron states. The cost of a memory transfer can exceed the cost of a floating-point multiplication operation by two orders of magnitude (Horowitz, 2014). TTFS-encoded SNNs fire at most one spike per neuron, thereby minimizing the energy cost in memory access.

Future research will be directed towards hardware implementation of this Time-To-First-Spike (TTFS) code, as well as the application on larger models and more challenging datasets. The massive reduction of spike counts in TTFS-encoded SNNs makes this model attractive for neuromorphic hardware platforms where energy costs are dominated by spike-induced memory fetches and spike routing.

17 Simulated for 32-bit floating-point in a Global Foundry 28 nm process.

## 2.3 TEMPORAL PATTERN CODE

Though inspired by the nervous system, deep ANNs employ a simplified neuron model that mimics the transfer function of non-leaky integrate-and-fire neurons in form of analog activation values. Even in the more biologically realistic class of SNNs, the predominant information transmission method is based on rate codes, which are potentially slow and inefficient due to redundant spikes. Temporal codes based on single spikes have been proposed but are difficult to scale up to applications in Deep Learning due to their sensitivity to noise in spike timing. Here we evaluate an encoding scheme based on temporal spike patterns, which inherits the efficiency of temporal codes but maintains the robustness of rate codes. This pattern code is evaluated on the MNIST, CIFAR-10, and ImageNet object recognition tasks. We compare against the performance of ANNs, rate-coded SNNs, and (where available) latency-coded SNNs, using the classification error and operation count as preliminary performance metrics. But since these models all use different elementary operations in hardware, we estimate the power consumption associated with the computational cost of each type of encoding, and take into account the effect of reduced-precision weights and activations. Finally, we explore a set of entropy codes that allow further compression due to the non-uniform distribution of neuronal activity observed in real-world tasks. The temporal pattern code achieves up to 42× reduction in the cost of operations at less than 1% increased error compared to the ANN. Compared to the rate-coded SNN, the error improves by 2% while the estimated power consumption decreases by 35×.[18]

### 2.3.1 Introduction

The question of neural coding - how signals are represented in the brain - has engaged neuroscientists at least since the place theory of pitch perception by Helmholtz in the 19th century. With the revival of ANNs in the late 1980's, interest in the principles of neural coding have spread to the machine learning community and inspired pioneering work such as Maass, 1997a,b. The subsequent success of DL (LeCun *et al.*, 2015) was in part made possible by the increased availability of computational power from hardware in the form of GPUs, but the limited resources available from platforms used in real-time mobile applications continue to fuel the search for more efficient ways to process DNNs. If not based on practical considerations alone, this research effort may further be motivated by the realization that our brain is able to accomplish complex cognitive tasks with the energy consumption of a light bulb (Mink *et al.*, 1981).

SNNs have emerged as viable candidates for more economical processing in DL Esser *et al.*, 2016. Taking inspiration from the biological nervous system, information in SNNs is typically transmitted in form of discrete events (spikes), rather than real values as in ANNs. This way, SNNs are able to use cheaper elementary operations than their analog counterparts (additions vs. MACs), and to exploit spatial and temporal sparsity, i. e. to skip redundant computations where neurons in the network are inactive (Yousefzadeh *et al.*, 2019). Whereas ANNs are operated in a synchronous,

18 The material in this section is currently under review for publication.

frame-wise manner, the processing mode of SNNs is asynchronous and driven by changes in the input. Such an event-based paradigm is attractive for low-latency, low-cost applications, which has been demonstrated e. g. in gesture recognition (Amir *et al.*, 2017), robotic target tracking (Rueckauer *et al.*, 2018), and object detection (Kim *et al.*, 2019b).

However, operating DNNs in the spiking regime has its challenges, starting with the problem of obtaining suitable weights for the synaptic connections. Directly training the SNN is difficult because of the non-differentiable nature of the spike generation mechanism. Employing surrogate gradients or a smoothed version of the activation function are promising ways to circumvent this problem (Lee *et al.*, 2016; Shrestha *et al.*, 2018), but direct training has not yet been shown to scale well to full-scale models like VGG-16 (Simonyan *et al.*, 2014b), GoogleNet (Szegedy *et al.*, 2015), or ResNet (He *et al.*, 2016). An alternate approach to spike-based training is conversion of an ANN, where one has a wealth of DL tools available to obtain high starting accuracy on arbitrarily large models. A drawback of this approach is that there is often a drop in accuracy when porting the model into the spike domain. Many of these conversion methods employ a form of rate-coding, where the average firing rate of a neuron in the SNN emulates the analog activation value in the corresponding neuron of the ANN. In a theoretical analysis, Rueckauer *et al.*, 2017 showed that such a rate code can approximate the ANN to arbitrary precision, but only in the limit of a large simulation duration.

Since then, several conversion schemes have been proposed to maintain high SNN accuracy at lower computational cost. Reduced spike-rates can be achieved e. g. by training the original ANN with a regularization term on the firing rates (Neil *et al.*, 2016). Unfortunately, low rates are more susceptible to approximation errors, which can be compensated by quantization-aware training (Sorbaro *et al.*, 2020), or by employing an adaptive neuron model (Zambrano *et al.*, 2016, 2019). A more rigorous approach is to forgo rate-codes altogether and encode ANN activation values in single spikes or short firing patterns, where information is contained in individual spike timing. Biologically inspired temporal coding in SNNs has a long history (see e. g. Bohte, 2004; Hanuschkin *et al.*, 2010; Maass, 1997c, 2015), but only recently have these principles been applied to multi-layer neural networks (Mostafa *et al.*, 2017; Orchard *et al.*, 2015c; Rueckauer *et al.*, 2018; Stöckl *et al.*, 2019, 2020). In Rueckauer *et al.*, 2018 (Sec. 2.2 in this thesis) for instance, real-valued ANN activations are encoded by the inverse TTFS (called *latency-coded* in the remainder of the Section). Every neuron fires at most one spike, which makes inference very efficient, but also very fragile towards noisy or delayed spikes.

In this Section we evaluate an encoding based on temporal firing patterns which we open-sourced as part of the "SNN toolbox" (Rueckauer *et al.*, 2017) mid 2017, and which has recently been formulated independently again (Stöckl *et al.*, 2020). This pattern code aims to find a middle ground between the fast but brittle temporal codes and the slow rate-codes. In our Temporal Pattern Code (TPC), individual spikes carry information in their timing, but rather than relying on a single spike per neuron we allow for subsequent spikes to supplement the message. To achieve this, the analog value to be encoded is transformed into a string of bits, for instance using the binary number format. This transformation introduces the notion of time:

each nonzero bit in the string represents a spike in a firing pattern. The amount of information carried by individual spikes in the pattern decreases from a spike at the position of the Most-Significant Bit (MSB) to the position of the Least-Significant Bit (LSB).

An immediate benefit of this encoding is the ability to trade off operations and latency against accuracy: By dropping some number of LSBs, we can sacrifice accuracy to save operations and speed up inference time. Exploiting low-precision in deep neural networks for efficient inference has been a topic of great interest (Choukroun *et al.*, 2019; Hubara *et al.*, 2016; Mishra *et al.*, 2018; Moons *et al.*, 2016; Rastegari *et al.*, 2016; Zhou *et al.*, 2017; Zhou *et al.*, 2016). In contrast to low-precision ANNs, where dedicated processing elements are required to perform operations at different levels of quantization, precision in our coding scheme can be varied dynamically using the same processing elements, by reducing the number of time steps per spike sequence. This tradeoff is investigated in Sec. 2.3.3.1, where we also take into account the reduced power consumption of low-precision ANNs for fair comparison.

With the motivation of improving energy efficiency by varying the precision of activations in individual layers, Judd *et al.*, 2017 developed an ANN hardware accelerator that may be seen as a possible implementation of the TPC code presented here. Their core idea is to replace the bit-parallel processing of the activation-weight product by bit-serial processing, which is equivalent to the temporal integration of spike patterns in TPC. While Judd *et al.*, 2017 investigate this processing paradigm in the context of accelerating ANNs, the present study adds to their work by providing the perspective of spike-based processing, in particular the comparison against rate- and latency-coded SNNs.

Aside from the binary number format (used in Judd *et al.*, 2017; Stöckl *et al.*, 2020), many other representations are possible for encoding ANN activations in form of spike patterns. In particular, a loss-less compression method like entropy coding assigns fewer bits (i. e. spikes) to values that appear more frequently, thus reducing the overall number of operations. We evaluate a set of compressed binary representations in Sec. 2.3.3.4, using as quality metric the Kullback-Leibler Divergence (KLD) between the original and the encoded activation value distribution. As expected from information theory, naive binary representation shows higher KLD than entropy-based encodings, with Huffman code minimizing the KLD, indicating optimal compression.

We evaluate pattern-coded SNNs on MNIST, CIFAR-10, and ImageNet, and compare against latency- and rate-encoded SNNs, as well as ANNs at varying numeric precision. Qualitatively, the pattern-encoded models reach ANN-equivalent accuracy with fewer operations than rate-coded models and more operations than latency-coded models, confirming our initial hypothesis. Further, TPC models scale better to more difficult tasks than both other spike encodings. Quantitatively, TPC requires up to 20× fewer operations than the ANN at equivalent accuracy. For ImageNet, we compare against low-precision ANNs and estimate a reduction of up to 42× in power consumption based on operation-count, using an 8-bit MobileNet (Howard *et al.*, 2017). Compared to the rate-coded SNN, TPC achieves 2% lower classification error with an estimated 35× reduction in power consumption.

### 2.3.2 Methods

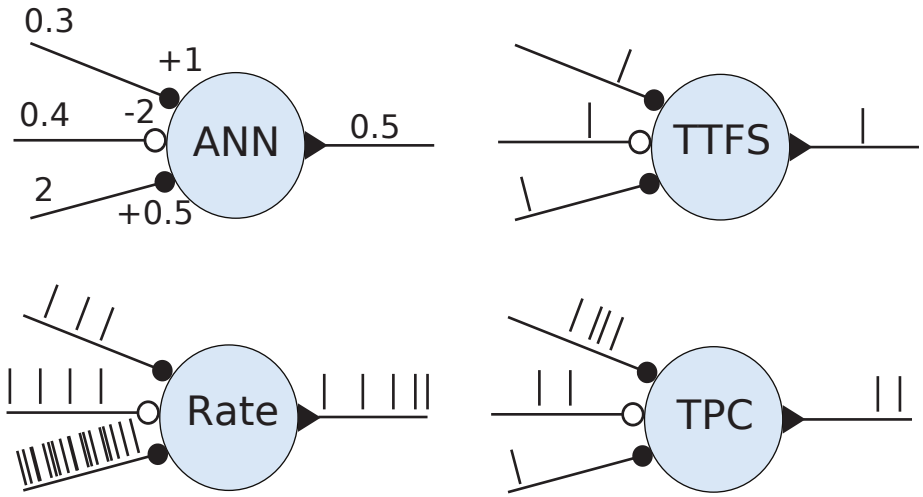


**Figure 2.10:** Illustration of the four DNN operating modes studied in this paper. Adapted from Liu *et al.*, 2019.

The basic principle underlying the temporal pattern encoding is that $n$-bit activation values in the ANN can be represented by a sequence of $n$ spikes in the SNN, e. g. using the binary number format. The time of spike in a pre-synaptic neuron $i$ determines the amount of charge deposited on the membrane potential of a post-synaptic neuron $j$. For instance, assume neuron $i$ has an activation value of 9, whose binary representation is 00001001 in an $n = 8$ bit word. Thus, neuron $i$ will fire only two spikes, the first after 5 time steps, and the second at time step 8. The post-synaptic neuron $j$ adds an amount of charge to its membrane potential that is proportional to the synaptic weight $w_{ij}$ as usual. However, to take into account the relative timing of spikes, this synaptic strength is modulated by a time-dependent kernel $\epsilon_{ij}(t)$.

This scheme can be formulated within the framework of the spike-response model (Gerstner *et al.*, 2014). We start from the general form of the membrane potential

$$u_i(t) = \sum_{j \in \Gamma_i} \sum_{t_j^{(f)} \in \mathcal{F}_j} w_{ij} \epsilon_{ij}(t - t_j^{(f)}) + I_{\text{ext}}(t) + \sum_{t_i^{(f)} \in \mathcal{F}_i} \eta_i(t - t_i^{(f)}), \tag{2.30}$$

where the kernel $\epsilon_{ij}$ represents the PSP caused by an input spike from one of the pre-synaptic neurons in $\Gamma_i$, and the external input current $I_{\text{ext}}(t) = b_i$ is given by the bias $b_i$. The term $\eta_i$ models the shape of the action potential and the refractory period, and is not used in this work. The time $t \in [0, n]$ is limited to the number of bits $n$, at the end of which the membrane potential is reset and a new sample (e. g. image frame) is presented for the next $n$ time steps. This time threshold $n$ replaces the voltage threshold that was used to trigger spikes in previous coding

schemes. When reaching the time threshold, the post-synaptic neuron $i$ fires a burst of spikes, whose times $\mathcal{F}_i = \{t_i^{(f)}\}$ correspond to the binary representation of the accumulated membrane potential. Such a time threshold could be implemented via a periodically firing "clock" neuron (see Sec. 2.3.4 for a discussion).

By choosing the synaptic kernel $\epsilon_{ij}(t - t_j) = \mathrm{H}(t - t_j) \cdot 2^{-t_j+n-1}$, with H the Heaviside step function, we arrive at our final expression for the membrane potential:

$$u_i(t < n) = \sum_{j \in \Gamma_i} \sum_{t_j^{(f)} \in \mathcal{F}_j} w_{ij} \cdot \mathrm{H}(t - t_j^{(f)}) \cdot 2^{-t_j^{(f)}+n-1} + b_i. \tag{2.31}$$

By construction, at $t = n$, the accumulated kernel contribution $\sum^{\mathcal{F}_j} 2^{-t_j^{(f)}+n-1}$ of pre-synaptic neuron $j$ equals exactly that neuron's membrane potential, $u_j(n)$. In the input layer, this value corresponds to the pixel activation $a_j$. Thus, membrane potentials in the first hidden layer are a perfect reflection of the corresponding activation values in the ANN[19]: $u_i(t = n) = \sum^{\Gamma_i} w_{ij} \cdot a_j + b_i = a_i$. The same argument extends iteratively to higher layers, which asserts that the encoding is loss-less if all $n$ timesteps are used during simulation. A lossy encoding is obtained by running for $k$ fewer timesteps, which is equivalent to truncating the last $k$ bits of the binary representation. The effect on accuracy and computational cost is evaluated in Sec. 2.3.3.1.

The proposed temporal pattern code was implemented using Python and Tensorflow, and is open-sourced as part of the SNN toolbox (Rueckauer *et al.*, 2017). This software package also includes tools for rate-based conversion and for encoding SNNs using the TTFS scheme. In Sec. 2.3.3.1 we apply all three of these encoding methods to convert SNNs and compare their performance against their original ANNs. A schematic illustration of the operation modes of analog and spiking neural networks is shown in Fig. 2.10.

### 2.3.3 Results

To evaluate the performance and robustness of TPC, we measure the classification error of models trained with 32 bit weights and activations, after conversion to TPC at different values of the bitwidth $n$ (representing the number of inference time steps per sample). The resulting error-operations tradeoff curve allows comparison against other spike-codes and the original ANN performance. We first present tradeoff results for MNIST, CIFAR-10 and ImageNet (Sec. 2.3.3.1), and then studies of the effect of training the ANN with low-precision weights and activations, and estimate the power consumption of the low-precision add, MAC, and shift operations needed for inference (Sec. 2.3.3.3, summary in Table 2.7). Finally, various loss-less compression schemes are explored as alternatives to the naive binary representation (Sec. 2.3.3.4).

19 The ReLU activation is assumed.

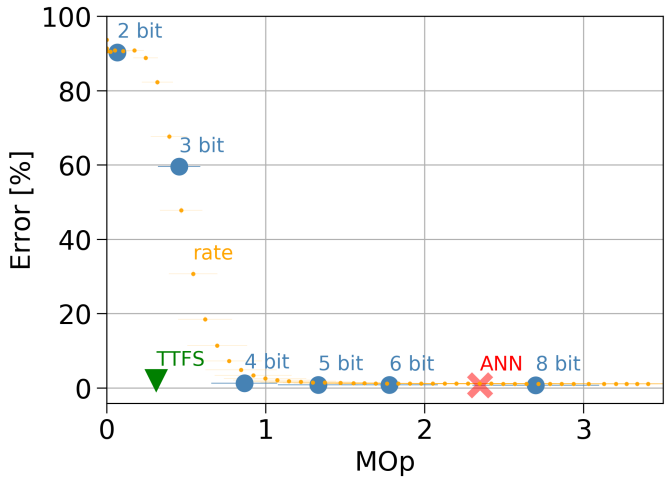


**Figure 2.11:** Classification error versus operation count per frame of the LeNet-5 ANN and the converted SNNs tested on MNIST. Three conversion methods are compared, based on rate (yellow), latency (green), and pattern encoding (blue). For the pattern code, each data point represents an experiment where the model was run for $n$ timesteps per test sample, which is equivalent to truncating the network activations to $n$ bit (as indicated by the label).

**Table 2.4:** Summary for MNIST.

| | Error [%] | Operations [M] |
|---|---|---|
| ANN | 0.72 | 2.35 |
| rate-SNN | 1.16 | 1.76 |
| TTFS | 2.0 | 0.31 |
| TPC 8 bit | 0.75 | 2.70 |
| TPC 6 bit | 0.76 | 1.78 |
| TPC 5 bit | 0.81 | 1.33 |
| TPC 4 bit | 1.26 | 0.87 |
| TPC 3 bit | 59.52 | 0.45 |
| TPC 2 bit | 90.26 | 0.07 |

#### 2.3.3.1 *Trading off Classification Error vs Operations*

**LENET-5 ON MNIST** As a proof-of-concept, we first evaluate the pattern code on the classic LeNet-5 architecture for MNIST (LeCun *et al.*, 1998). We compared three spike encoding methods. As shown in Fig. 2.11, both the rate and pattern codes perform similarly, with the classification error of the TPC network converging slightly faster to the error rate of the ANN. Specifically, TPC at $n = 4$ bit requires about 1.8× fewer operations than the rate code and 2.7× fewer operations than the ANN to achieve the same error rate. Compared to TPC at 4 bit, TTFS is another 4× more efficient in this experiment because only a single spike is fired per active

neuron. However, this latency code suffers from unacceptable accuracy loss in the more difficult tasks studied below. The results are given in Table 2.4.

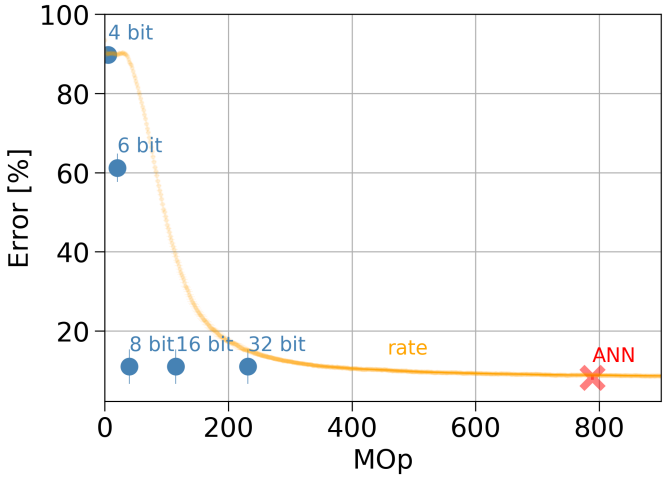


**Figure 2.12:** Classification error versus operation count per frame of the original ANN and the converted SNNs tested on CIFAR.

**MOBILENET ON CIFAR-10** For CIFAR-10 (Krizhevsky, 2009) we choose the MobileNet-V1 (Howard *et al.*, 2017) architecture, which achieves 8.82% classification error with comparably small parameter footprint due to the use of depthwise-separable convolutions. As seen in Fig. 2.12, TPC needs about 14× less operations than rate code and 20× less than the ANN at iso-accuracy. The quantitative results can be found in Table 2.5.

**Table 2.5:** Summary for CIFAR-10.

| | Error [%] | Operations [M] |
|---|---|---|
| ANN | 8.82 | 789 |
| rate-SNN | 9.4 | 576 |
| TPC 32 bit | 8.82 | 232 |
| TPC 16 bit | 8.82 | 115 |
| TPC 8 bit | 8.82 | 39 |
| TPC 6 bit | 61.19 | 20 |
| TPC 4 bit | 89.73 | 5 |

**MOBILENET ON IMAGENET** For ImageNet (Russakovsky *et al.*, 2015), we use the same architecture as for CIFAR-10, except that the resolution of the input images is higher (256 × 256 instead of 32 × 32). The ANN achieves a top-1 error of 30.41% at 9.66 GOp per frame (Fig. 2.13). The rate-encoded SNN fails to reach this error rate

within the same computational budget. [20] The pattern-encoded network reaches the same error as the ANN when run for $n = 32$ time steps per frame. Even though each analog activation value is replaced by a pattern of up to 32 spikes, the TPC model still requires 1.3× fewer operations than the ANN, because TPC natively skips over zeros in the feature maps.[21]. When the network runs for $n = 16$ time steps, TPC has to process fewer spikes, leading to 2.7× fewer operations. The error increases by 0.72%, which is equivalent to the error obtained when naively rounding the ANN activations to 16 bits. Reducing run time further to $n = 8$ timesteps incurs a more substantial drop of 8.9%. This increased error can be avoided by training the ANN with 8-bit activations, which is described in more detail in Sec. 2.3.3.3. The ImageNet results are summarized in Table 2.6.

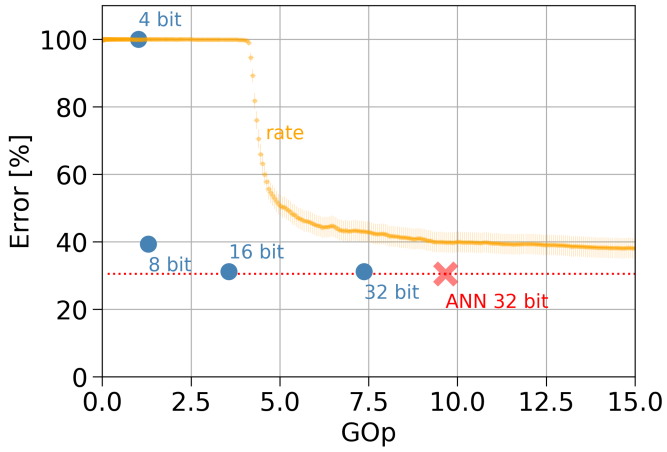


**Figure 2.13:** Classification error versus operation count per frame of the original ANN and the converted SNNs tested on ImageNet.

**Table 2.6:** Summary for ImageNet.

| | Error [%] | Operations [G] |
|---|---|---|
| ANN | 30.41 | 9.66 |
| rate-SNN | 36.01 | 23.81 |
| TPC 32 bit | 30.41 | 7.37 |
| TPC 16 bit | 31.13 | 3.56 |
| TPC 8 bit | 39.30 | 1.29 |
| TPC 4 bit | 100 | 1.02 |

20 Inspection of the parameter distribution revealed weight and bias values several orders of magnitude larger than the standard deviation. Rueckauer *et al.*, 2017 showed that such outliers can cause erroneous spikes at the onset of the simulation while spikerates are still ramping up, which delays convergence to the desired accuracy. Removing these outliers by training the model with hard constraints or regularization would likely improve the rate-based conversion but was not within the scope of the present work.

21 This advantage would be evened out in a dedicated zero-skipping hardware (Aimar *et al.*, 2019) or algorithm (Messikommer *et al.*, 2020)

#### 2.3.3.2 *One-Hot Encoding*

The efficiency of this pattern code can be further improved by quantizing activations to powers of 2. The binary pattern representation then becomes maximally sparse. We did not train the model ourselves to achieve equivalent accuracy at power-2 quantized activations, which has been shown in other work (Gudovskiy *et al.*, 2017). However, in a preliminary experiment, we found that MobileNet on ImageNet would require 0.6 GOp in a power-2 encoding, a reduction of 16×.

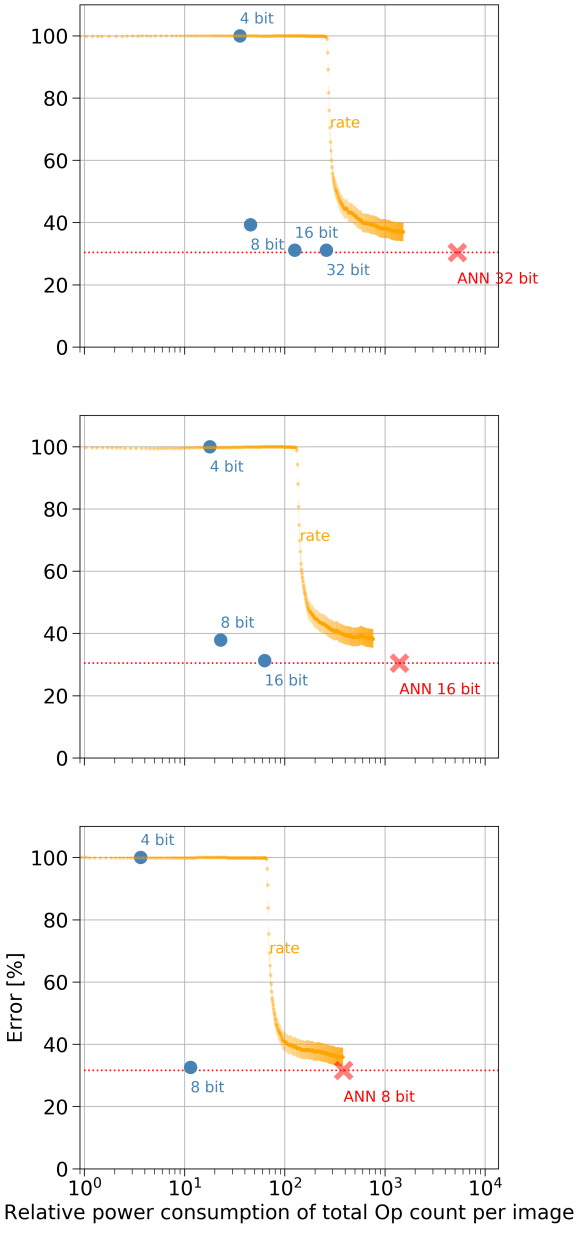


**Figure 2.14:** Classification error versus relative power consumption of the original ANN and the converted SNNs tested on ImageNet. Each of the three panels represents the same conversion experiment but starting from an ANN trained with 32 (top), 16 (center), and 8 bit activations (bottom).

**Table 2.7:** Summary of power estimates for ImageNet (power is measured relative to the cost of one adder). The factor in parenthesis represents the reduction relative to the ANN.

| ANN | 32 bit | | 16 bit | | 8 bit | |
|---|---|---|---|---|---|---|
| | Error [%] | Power | Error [%] | Power | Error [%] | Power |
| ANN | 30.41 | 5255 | 30.47 | 1391 | 31.58 | 386 |
| rate-SNN | 36.01 | 1526 (3×) | 37.07 | 764 (2×) | 34.88 | 381 (1×) |
| TPC 32 bit | 30.41 | 259 (20×) | | | | |
| TPC 16 bit | 31.13 | 125 (42×) | 31.28 | 63 (22×) | | |
| TPC 8 bit | 39.30 | 45 (117×) | 37.84 | 23 (60×) | 32.59 | 11 (35×) |

#### 2.3.3.3 *Estimated Power Consumption*

One inherent problem with the comparison in Sec. 2.3.3.1 is that we simply counted the raw number of operations, irrespective of their type. Standard ANNs perform MACs operations, TPC networks require addition and bit-shift operations, and rate- and latency-coded SNNs perform only additions. To account for the different cost associated with each type of basic operation, we repeated the tradeoff analysis for ImageNet, this time weighting each operation with its respective power consumption in hardware. Instead of raw operation count, we then report the power which is consumed to perform the operations required to run inference on one frame.

This metric is calculated as follows:

$$C = \sum_{\text{op}}^{\{\text{Op types}\}} c_{\text{op}} N_{\text{op}}, \tag{2.32}$$

where {Op types} is the set of operation types present in the considered models (MAC, addition, bitshift), $c_{\text{op}}$ is the cost associated with a particular op, and $N_{\text{op}}$ the number of times it is performed during inference of one sample.

The numbers $c_{\text{op}}$ represent the power consumption relative to one adder. For two operands of bitwidth $b_1$ and $b_2$, the cost of addition follows $c_{\text{add}} = \max(b_1, b_2) + \frac{|b_1 - b_2|}{2}$; a MAC consumes $c_{\text{MAC}} = b_1 \cdot b_2$. To shift a $b_1$ bit word by $b_2$ possible bits would consume $\frac{2}{3} b_1$ relative to the adder. However, in our TPC models we only need to shift by 0 or 1 bits as the bitstring is processed serially, which requires $c_{\text{shift}} = \frac{b_1}{5}$. These power estimates for $c_{\text{op}}$ were validated on a Globalfoundries 28 nm SLP process, using the synthesis tool Synopsys Design Compiler 2018.9.

Since the cost of an operation depends on the bitwidth of the operands, we consider three scenarios: The first ANN is trained with 32-bit activations, the second with 16-bit, the third with 8-bit. The bitwidth of the weights is fixed to 8 bit in all models to match the lowest bitwidth considered for the activations. Starting with these three ANNs, we again perform the conversion to rate- and pattern-encoded SNNs and measure the tradeoff curves (Fig. 2.14). When converted from the 16 bit

ANN, the TPC model achieves a relative reduction of power consumption up to 42× with less than 1% increased error (c. f. Table 2.7).

From the three panels in Fig. 2.14 we observe the following trend: As we move from 32 bits (top plot) to 8 bits (bottom plot), all data points shift towards the origin, i. e. to the desired regime of lower error rate at lower power consumption. To dissect this result in more detail, consider first the error rate. In case of the ANN, quantization-aware training is able to maintain the accuracy of the 32-bit model even when going to 16 and 8 bit. In case of the rate-coded SNN, training with reduced precision appears to make the model more robust to approximation errors that are bound to occur in a network of this size. In case of the pattern-coded model, training the ANN at low precision prepares the converted SNN to handle truncation of LSBs when reducing the number of time steps during inference. In terms of power consumption, we expect a shift to the left for two reasons: (1) The shorter bitwidth of operands results in a lower cost per operation. (2) Reduced precision is often accompanied by an increase in sparsity, which would reduce the total number of operations. However, the latter did not play a role here; we found that the sparsity did not change significantly. While this power analysis enables a comparison of ANNs and SNNs purely based on the cost of their elementary operations, it does not take into account some important aspects of a potential implementation, for instance the cost associated with memory traffic. Reading a bit string of weights from external Dynamic Random Access Memory (DRAM) costs as much as 21 pJ per bit in an LPDDR3 memory model (Schaffner *et al.*, 2015), seven times as much as a 32 bit multiplication (3.1 pJ according to Horowitz, 2014). In rate-coded stateful SNNs, such fetches are more frequent and less predictable than in ANNs. Because TPC-coded networks are stateless and evaluated in a layerwise fashion, they share the reduced memory traffic of ANNs. The related hardware implementation by Judd *et al.*, 2017 has demonstrated about 2× speedup and 57% improved energy efficiency over the DaDianNao ANN accelerator (Chen *et al.*, 2015) using models of comparable size as the MobileNets considered here. Thus, when taking into account memory traffic, we can expect the performance difference between rate- and pattern-coded SNNs to widen further.

When implemented on dedicated neuromorphic hardware (Davies *et al.*, 2018; Merolla *et al.*, 2014), the power consumption of rate-coded SNNs due to off-chip memory traffic is partially alleviated by storing weights and states in area-expensive SRAM, which lowers the energy consumption for reads by two orders of magnitude compared to DRAM. Still, the large amount of spike events in rate-coded models may cause spike congestion in chip-to-chip traffic (internal results), increasing latency during inference. This potential bottleneck would be mitigated by a finite pattern code as presented here.

#### 2.3.3.4 *Compression*

In the experiments above, we rely on the binary number format to represent ANN activations. While this simple block code is effective, variable-length symbol codes can potentially be more efficient by assigning fewer bits to more frequent values. For instance, Huffman code, a form of entropy encoding, generates codewords with a length that is proportional to the negative log likelihood of the occurrence

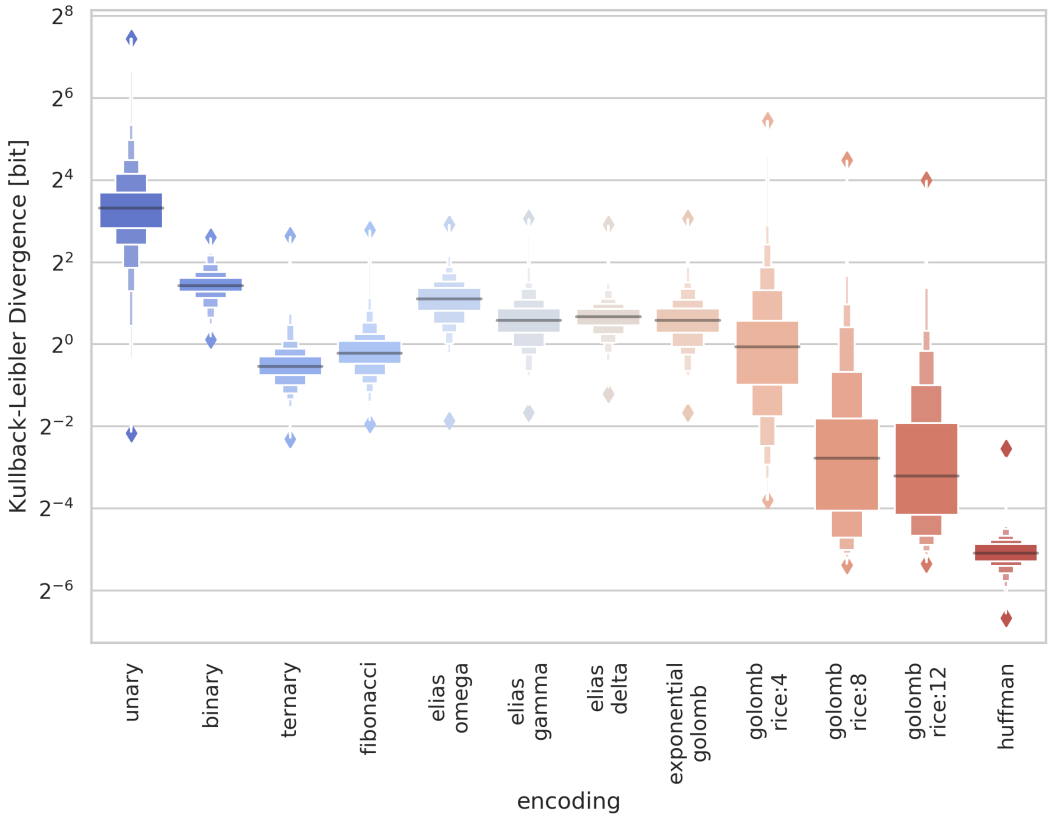


**Figure 2.15:** Kullback-Leibler Divergence from the intrinsic distribution of 12 different codes to the empirical distribution of activations measured in MNIST, CIFAR-10 and ImageNet tasks, using LeNet-5, ResNet, MobileNet, VGG-16 and Inception-v3.

of that particular codeword. Thus, to construct the codebook, Huffman coding requires knowledge of the underlying probability distribution, and is relatively costly to compute. In contrast, universal codes can be used without knowing the exact distribution of values to encode, which makes universal codes cheaper but also less compressive. Among universal codes, Rice, Golomb, and Elias codes are known to be highly suitable for geometric distributions (Gallager *et al.*, 1975), where the occurrence of small values in the input is significantly more likely than large values.

To quantify which of these various codes are good candidates to replace the binary block code in our TPC method, we determined the Kullback-Leibler Divergence (KLD) of the activation distribution $p$ found in practice from the optimal distribution $q$ for a particular encoding. This measure can be interpreted as the additional number of bits needed to encode activation values in $p$ using the code optimized for $q$. First, we obtained the distribution $p$ of hidden layer activations by running inference on the entire test set of the MNIST, CIFAR-10 and ImageNet tasks, using LeNet-5, ResNet, MobileNet, VGG-16 and Inception-v3 architectures. Next, we determined the underlying distribution $q$ of 12 different encoding schemes, including unary, binary, ternary, Fibonacci, Huffman, as well as variants of Elias, Golomb and Rice codes. As discussed before, the underlying distribution of some of these

codes is independent of the distribution of values to be encoded. For instance, the distribution for the binary number format is flat at $2^{-n}$ (where $n$ is the bitwidth), indicating that the encoding is optimal if all values appear equally likely. The distribution underlying unary code decays exponentially with $2^{-k}$ ($k$ covering the domain of values to be encoded); the ternary distribution follows a power law $k^{-1.26}$. Huffman code on the other hand depends on the distribution of encoded values, so we computed the Huffman code book for each of the measured activation distributions. Given the word lengths $L$ in the code book, the distribution for Huffman is given by $2^{-L}$ (for the compression to be optimal, a word should be less frequent the longer it is.).

Having determined both the activation distribution $p(k)$ as well as the inherent code distribution $q(k)$, we can then compute the KLD as

$$\mathrm{KLD} = \sum_k p(k) \log \frac{p(k)}{q(k)}. \tag{2.33}$$

As the KLD measures the dissimilarity between two distributions, we can use it as a proxy for the suitability of a given code to represent the activity distribution in a given task and network. Figure 2.15 shows that for the range of tasks tested here, Huffman code indeed fits the activation distribution optimally as expected. The binary code is found at the other end of the spectrum among the worst encoding options. A block code apparently does not fit the requirements of ReLU activation values, whose distribution tends to peak strongly at small values and then decay with a long tail. Promising candidates are found among the family of Golomb-Rice codes, which fall short of Huffman in terms of compression but are less expensive to compute. An interesting question for future work will be to evaluate the feasibility of these codes in custom hardware. Since we must expect an encoding / decoding overhead when using variable-length symbol codes, a plain binary code may turn out to be more favorable than what Fig. 2.15 suggests.

### 2.3.4 Discussion

From the perspective of SNN conversion methods, TPC offers a number of advantages over the rate- and latency-based encodings. First, the magnitude of spike rates is decoupled from the magnitude of activations. Whereas high activations in the ANN result in high spike rates in the SNN under rate coding, a pattern code may use the same number of spikes for a low as for a high activation value. This attribute is beneficial because low activations are problematic in both rate and TTFS code due to their susceptibility to noise.

Second, the time-encoding of activation values applies a log-compression. In the TTFS code, reducing the activation by a factor $2^k$ slowed down the spike by $2^k$. Here, the delay would be $k$ time steps. For instance, a common activation value of $0.016 \approx 2^{-6}$ would require 64 timesteps to be represented in our TTFS code but only 6 steps in the pattern code.

Third, when using the full bitwidth ($n = 32$), TPC does not involve any approximation errors as in the rate and TTFS code. Each layer in the SNN is guaranteed to reach a perfect representation of the ANN within a finite simulation time.

Fourth, the upper limit for the required simulation duration is given by the bit precision, i. e. at most 32 time steps, which may be up to 2 orders of magnitude shorter than in a standard rate code.

Fifth, with a shorter simulation duration, we can expect the spike rates and thereby the operations and memory traffic to be lower than in a rate code.

Sixth, with layers being processed sequentially, only the neuron state variables of the current layer need to be kept in memory. In a rate code, neuron states for the whole network need to be stored and are accessed repeatedly in an unpredictable manner.

Seventh, given a model with bit-precision $n$, an accuracy-latency trade-off curve can be obtained simply by reducing the simulation duration per layer to values $T < n$. The resulting SNN accuracy will be equivalent to an ANN where the $n - T$ LSBs of the activation values are clipped.

There remain open questions about the proposed method, in particular with regard to biological plausibility. First, Unlike the asynchronous processing paradigm of SNNs, the pattern code treats one layer at a time for $n$ time steps before moving on to the next. Thus, the pattern code sacrifices pseudo-simultaneity, i. e. the ability of the network to produce a preliminary guess at the beginning of a simulation and then refine it as more data comes in. Input samples would of course be pipelined for an efficient use of the whole network. However, there will be a delay of at most $n$ time steps to aggregate one sample.

Second, the encoding mechanism relies on a time threshold or clock neuron to trigger spikes. However, most other coding schemes employ a similar mechanism to cause a global time or membrane potential reset between samples in an uncorrelated input sequence, one example being the rank-order models of communication in the visual pathway (VanRullen *et al.*, 2001). Neuroscientists have suggested visual saccades or micro-saccades as possible implementation of this mechanism (Martinez-Conde *et al.*, 2000; Rodieck, 1998).

Third, it is not clear how real neurons would convert a given analog value into a particular spike pattern. One possibility would be to combine a population code with a rank-order or latency code: Neurons in a population may respond to a stimulus with different delays; the first spikes could constitute the temporal spike pattern (bit sequence in our TPC), with their respective order determining their efficacy on the post-synaptic neuron (c. f. (VanRullen *et al.*, 2001) for a candidate for such a rank code.). Further evidence for information to be encoded in spike patterns has been presented in Victor, 2000.

#### 2.3.4.1 *Related Work*

A recent pre-print (Stöckl *et al.*, 2020) proposes a very similar method to TPC[22]. The main algorithmic difference is that they employ an exponentially decreasing firing threshold instead of a time threshold to generate the spike pattern, which does not lead to a different bit transmission than used here, but may be more biologically realistic. Their reported SNN error rate of 7.58% for ResNet50 on CIFAR-10 is on par with the ANN (7.01%) when using $n = 10$ time steps, likewise with ImageNet

22 It postdates our work, which has been made open-source as part of the SNN conversion toolbox (Rueckauer *et al.*, 2017) mid 2017.

(24.9% in SNN vs. 24.78% in ANN, using $n = 20$ time steps). These SNN error rates are better than the ones in our work, as we test the smaller MobileNet architecture (28 layers, 5M neurons, and 4M parameters vs. 50 layers, 9.6M neurons, and 26M parameters in ResNet50). A more detailed quantitative comparison between the two works is not possible as the authors provide no performance metric besides the accuracy and spike count; the effects of varying $n$ or the bitwidth of the original ANN is also not reported.

Zhang *et al.*, 2018 likewise use the binary representation of activation values to generate spike patterns in the SNN. In their encoding, the number of spikes in the pattern grows logarithmically with the magnitude of the activation value. The method induces an accuracy loss during conversion that needs to be compensated for by training the ANN with a special objective function, non-linearity, and additional hyperparameters. Their results are limited to the MNIST task, where they achieve loss-less conversion (0.59% classification error) at 2.29 MOp.

The work by Judd *et al.*, 2017 replaces the bit-parallel processing of the activation-weight product by a bit-serial implementation, which is a way to integrate the spike patterns in the encoding discussed here. To make up for the increased number of clock cycles needed when going from bit-parallel to bit-serial, the parallelism inherent in CNNs is exploited. The authors report roughly 2× improved performance over a then state-of-the-art accelerator (Chen *et al.*, 2015). The proposed accelerator is 57% more energy efficient than the baseline at a cost of 32% additional area.

The work of Georgiadis, 2019 combines sparsification, quantization and entropy encoding of feature maps to achieve an acceleration at inference of up to 6× on Inception-v3 and MobileNet-v1 architectures. They further demonstrate that training the network with activity regularization to increase sparsity is beneficial for entropy encoding because the activity distribution will concentrate most of its mass around zero to then decay with a long tail. Thus, their pipeline of training the ANN with sparsity constraints and quantization of activations could be used complementary to our spike pattern representation, using a variable-length Golomb-Rice code as indicated in Sec. 2.3.3.4.

In the context of quantizing feature maps, Park *et al.*, 2017 investigate the benefits of choosing the bitwidth on a per-layer basis. The best results are obtained with *convex quantization*, where top and bottom layers have higher bitwidth than layers in the middle. Taking a step away from heuristic quantization rules, Wang *et al.*, 2019 automatically determines optimal layer-wise quantization policies using reinforcement learning with feedback from the target hardware accelerator, with a cost function that takes into account model size, latency, and energy, rather than proxy metrics like Floating-point operations (Flops) and memory accesses.

While our approach focuses on efficient inference, Jain *et al.*, 2018 make use of feature map compression to accelerate training of DNNs. Realizing that activation values obtained during forward pass have to be buffered for later use during the backward pass, the authors propose storing a compressed version of these feature maps and observe a 2× reduction of the memory footprint with a mere 4% performance overhead.

### 2.3.5 Conclusion

We evaluate an encoding scheme for spike signals in SNNs based on the binary number format. The proposed method addresses several shortcomings of rate- and latency-based encoding methods of SNNs, in particular poor scalability due to approximation errors, as well as the high number of redundant spikes in rate coding.

The presented method was evaluated on MNIST, CIFAR-10, and ImageNet, and improves on the classification error of rate-coded SNNs by 2% while reducing the estimated cost of operations by 35×. We evaluated further optimizations such as training the original ANN with reduced precision, or with a power-2 encoding of activations. Finally, we showed that variable-length coding schemes like Golomb-Rice are promising replacements for naive binary representation, in order to compress activation values optimally based on their empirical distribution.

# 3 EVENT-BASED PROCESSING ON NEUROMORPHIC HARDWARE

## 3.1 INTRODUCTION

The widespread success of DL has been accompanied by the development of easy-to-use tools like Keras (Chollet, 2015) and Pytorch (Paszke *et al.*, 2019) to tackle large-scale workloads with little overhead. Still, training and running DNNs on conventional hardware is often costly in terms of time and power consumption. This realization drives research efforts to develop more efficient algorithms (e. g. Blalock *et al.*, 2020) and dedicated hardware (e. g. Reuther *et al.*, 2019). SNNs, a particular class of such algorithms with the promise of efficient processing, draw inspiration from the brain by transmitting information asynchronously in form of discrete spikes. Neuromorphic platforms like SpiNNaker (Furber *et al.*, 2013), TrueNorth (Merolla *et al.*, 2014), and Loihi (Davies *et al.*, 2018) optimize for this kind of event-based computation and have demonstrated the potential to run neural networks at low latency and low energy consumption (Esser *et al.*, 2016; Stromatias *et al.*, 2015). However, this special-purpose hardware is currently difficult to use compared to Central Processing Units (CPUs) or GPUs, as it requires an intricate understanding of the low-level implementation details and hardware constraints. The present work aims to reduce this obstacle by introducing NxTF - an interface and compiler for DNNs on the neuromorphic chip Loihi.

Most SNN hardware platforms come with some form of high-level programming interface that allow users to specify neural networks, and a compiler that solves the problem of mapping the network graph onto the hardware substrate. Some examples include Spynnaker (Rhodes *et al.*, 2018), SpiNNTools Rowley *et al.*, 2019, TrueNorth Corelets (Sawada *et al.*, 2016), and the Loihi toolchain (Lin *et al.*, 2018). The present work differs from these tools by targeting specifically DNN models. By trading off generality, we are able to exploit the characteristic structure of DNNs for a more efficient use of hardware resources.

We begin this chapter by outlining in Sec. 3.2.1 the typical workflow of using our NxTF framework to deploy a DNN on Loihi. After reviewing the necessary backround about the Loihi architecture in Sec. 3.2.2, we explain the network compilation procedure in Sec. 3.2.3. Finally, we present two common use cases of NxTF in Sec. 3.3: Deploying on Loihi an SNN that A) has been trained directly on a spike-based dataset using SLAYER (Shrestha *et al.*, 2018), and B) that has been converted from an ANN trained on frames. We describe tools to benchmark the performance of these networks and compare their power consumption and execution time against results on other neuromorphic hardware, using the Energy Delay Product (EDP) as combined metric.

## 3.2 METHODS

As our core contribution we present here a user interface and compiler for DNNs on Loihi, called "NxTF" - the name denoting a bridge from Tensorflow to Loihi's NxSDK[1]. In the design of this compiler we followed two objectives: (1) To provide an easy-to-use tool that "feels" like Keras. (2) To optimize the allocation of resources on the neurocores and exploit Loihi's connection sharing features for efficient deployment of large-scale CNNs. Covering the first objective, we will first provide a high-level overview of the tool chain in Sec. 3.2.1. We will then introduce basic concepts of the Loihi hardware architecture to understand resource constraints and opportunities for resource sharing taken into account by the compiler. With this background we can then dive more deeply into the compilation method in Sec. 3.2.3 to show how we solve the second objective.

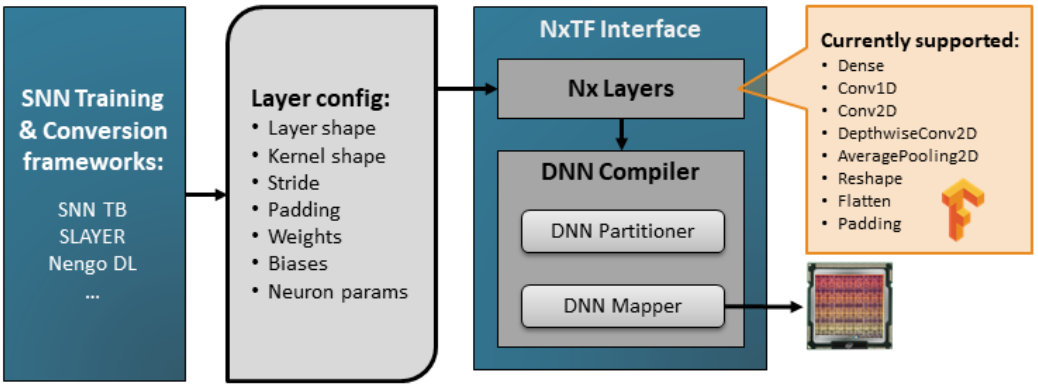


**Figure 3.1:** Overview of workflow. Starting from a trained or converted SNN, the user defines a model in Python using the Keras-like NxTF interface. The network is then partitioned and mapped onto Loihi by the DNN compiler.

### 3.2.1 Overview of Workflow

If a researcher wants to run a DNN on Loihi using our tools, the typical workflow might roughly follow the stages outlined in Fig. 3.1. The first step, obtaining a Spiking Neural Network (SNN), lies in the responsibility of the user; they may have already trained a model directly on spikes or utilized some third-party tool like Nengo (Bekolay *et al.*, 2014), SLAYER (Shrestha *et al.*, 2018), or the SNN toolbox (SNN TB) (Rueckauer *et al.*, 2017). The expected output at this stage (c. f. second box in Fig. 3.1) is a set of learned weights for each layer, the layer specifications defining e. g. the padding and stride of convolutions, and the desired neuron properties (like voltage decay and threshold value).

Given this information, the user can then define a network in Python using the NxTF interface. This interface is derived from Keras and follows the same syntax for setting up a Keras model, except that additional Loihi arguments (for threshold, time constants etc.) are supported. See Fig. 3.2 for a minimal code sample. The

1 "Nx" stands for the different generations of Loihi.

```python
from tensorflow.keras.layers import Input, Conv2D
from tensorflow.keras.models import Model

input_layer = Input((32, 32, 3))
output_layer = Conv2D(filters=64,
                      kernel_size=(3, 3),
                      activation='relu'
                      )(input_layer)
model = Model(input_layer, output_layer)
```

```python
from nxsdk_modules_ncl.dnn.src.dnn_layers \
    import NxInputLayer, NxConv2D, NxModel

input_layer = NxInputLayer((32, 32, 3), vThMant=255)
output_layer = NxConv2D(filters=64,
                        kernel_size=(3, 3),
                        vThMant=255
                        )(input_layer)
model = NxModel(input_layer, output_layer)
```

**Figure 3.2:** Comparison of model construction using `tf.keras` (left) and NxTF (right). The main difference is that class names get an `Nx` prefix and accept additional Loihi arguments (e. g. for the voltage threshold in this example).

result at this stage is an instance of NxModel, which inherits all functionality of its Keras base class.

To prepare the NxModel for deployment on Loihi, the user then calls the `compile` method, which consists of two parts: A partitioner and a mapper. The partitioner's job is to find the optimal distribution of the network across neurocores, whereas the mapper is responsible for bit-level configuration of Loihi registers.

Once the mapping is complete, the user can start up the board, inject input, run the model, and retrieve the output for evaluation. A detailed tutorial covering these steps is provided within the Intel Neuromorphic Research Community (INRC) framework[2]. In Sec. 3.3 we apply this pipeline on models trained directly on spike-based datasets as well as on models converted from frame-based ANNs. But before going to the results we will take a closer look at the Loihi hardware and NxTF compiler.

### 3.2.2 Loihi

#### 3.2.2.1 *Architecture Overview*

Intel's research chip Loihi is a neuromorphic processor optimized for the kind of asynchronous computation occurring in Spiking Neural Networks. Fabricated in Intel's standard 14 nm CMOS process, Loihi consists of 128 neurocores, each of which supports up to 1024 neurons. Three embedded x86 processors per chip enable off-chip data encoding and interaction with the neurocores. With its inter-chip communication interfaces, Loihi has been scaled up to various form factors, ranging from 2-chip USB device *Kapoho Bay* to the 768-chip rack *Pohoiki Springs* (Frady *et al.*, 2020).

Neurons on Loihi support a wide range of features, including synaptic plasticity, variable numeric precision of synaptic weights up to 9 bits, multi-compartment models, threshold adaptation for homeostasis, and configurable synaptic, axon and refractory time constants. To support hierarchical and repeated connectivity as in CNNs, Loihi provides a connection-sharing mechanism (described in more detail in Sec. 3.2.3).

2 https://github.com/intel-nrc-ecosystem/models, accessed Aug 2020

Unlike conventional synchronous architectures with globally clocked time steps, Loihi enables a flexible clock duration via a barrier synchronization mechanism. Cores process spike messages asynchronously and send blocking barrier handshakes to their neighbors to signal the completion of an algorithmic time step. The execution duration of a time step thus decreases with smaller spike loads within the core mesh. The resulting benefits for latency and power consumption should become especially evident in temporally coded SNNs that display a high degree of spatial and temporal sparsity (Mostafa *et al.*, 2017; Rueckauer *et al.*, 2018).

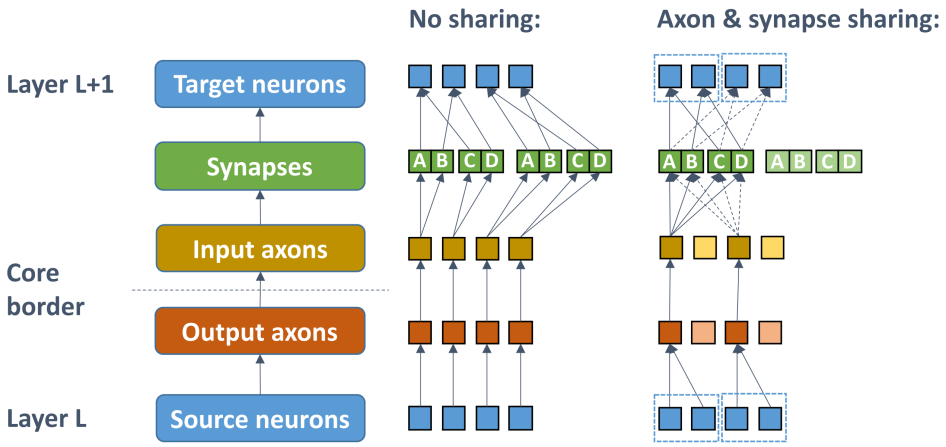


**Figure 3.3:** Fundamental components of a neurocore in Loihi. Left and center: A spike generated by a source neuron leaves the core via one or more output axons and enters another core through a corresponding input axon. The input axon contains a reference to synapses, which weigh the spike before it is accumulated on the membrane potential of its target neurons. Right: Axons and synapses are limited in number but can be shared to save resources on core.

#### 3.2.2.2 *Information Flow in Loihi Cores*

To understand the resource constraints on Loihi cores as well as opportunities for resource sharing within the compiler, we will briefly review the logical entities present on a neurocore (c. f. left column of Fig. 3.3). The central building blocks are compartments, which may form single- or multi-compartment neurons. For simplicity we will use the term "neuron" in this section to denote a single compartment.[3] The role of neurons is to accumulate spikes and generate a spike themselves once a threshold is crossed. Spikes are routed from a neuron to their destination via axons. In line with the hardware implementation we distinguish between output and input axons. An output axon is responsible for core-to-core connectivity; it routes spikes to a corresponding input axon on the same or another core. The input axon references a list of synapses, which connect the axon to its destination neurons. Spikes travelling through the axon are weighted by the respective synaptic strength before accumulation on the neuron's membrane potential. While this level of abstraction is sufficient for the purpose of the present work, a more detailed description can be found in Lin *et al.*, 2018.

3 A two-compartment neuron will be introduced in Sec. 3.2.3.6.

**Table 3.1:** Resource constraints on a Loihi chip. Additional limits (not listed) pertain to the compressed way synapses are stored on chip.

| Resource | Availability |
|---|---|
| Neurons | 1024 / core |
| Input axons | 4096 / core |
| Output axons | 4096 / core |
| Synapses | 128k / core |
| Cores | 128 / chip |

The availability of these entities (neurons, axons, synapses) on a given neurocore is limited (c. f. Table 3.1). Fortunately, Loihi provides a way to share axonal and synaptic resources (Fig. 3.3 right). In particular, a population of source neurons (dashed box in Fig.3.3) may use the same output axon (dark red square) to connect to a shared input axon (dark yellow square). Synapse sharing is realized if an axon points to the same physically stored synapse group than other axons (dashed arrows) rather than pointing to a copy of those synapses. Exploiting this resource sharing while adhering to core constraints is the aim of the NxTF compiler, which we address next.

### 3.2.3 Compilation Method

Given a network with certain topology, the purpose of the compiler is to find a distribution of neurons across multiple neurocores that makes optimal use of the available resources on chip. Because of their connectivity, layers cannot be partitioned independently of each other. On the other hand, partitioning all layers simultaneously is unfeasible because the combinatorial space of possible partitions is large. Instead, the compiler first proposes a given number of partition candidates per layer and then traverses the layer hierarchy to select the optimal subset. To quantify the notion of optimum, we define a cost function that takes into account hard constraints (e. g. number of available neurons per core) as well as soft constraints (e. g. the total number of cores used.). To understand the terms contributing to the total cost, we need to compare opportunities for resource sharing in DNNs and on Loihi.

#### 3.2.3.1 *Connection Sharing*

Deep Neural Networks are characterized by a regular connectivity and repeated architectural patterns. This fact can be exploited to develop a compiler optimized specifically for DL applications. Commonly seen *convolution* layers are particularly advantageous because of their efficient use of trainable parameters. For a given layer, a single set of weight kernels is re-used at every spatial location of the feature map. Fig. 3.4 illustrates this connectivity pattern within a CNN and defines basic terminology used throughout this section.

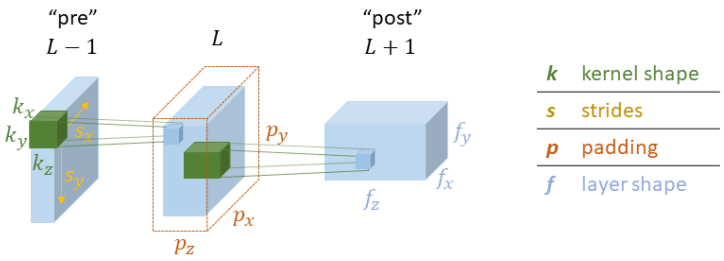


**Figure 3.4:** Connectivity pattern of a CNN. One set of kernels (green block) is shared among neurons of a given layer, and applied by striding (yellow arrows) along the height and width of the feature map (blue box). The layer may optionally be padded with zeros to maintain the same dimensions after convolution.

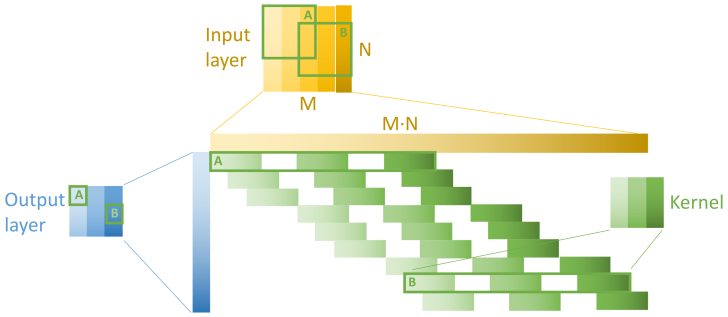


**Figure 3.5:** Unrolled connectivity matrix of a convolution layer.

NxTF flattens all layers to a 1D topology on Loihi, so we introduce another way of inspecting the CNN structure in Fig. 3.5. For clarity we consider only the spatial $(x, y)$ dimensions of a feature map, neglecting the channel $(z)$ dimension. The graphic illustrates the convolutional connectivity matrix from the input layer (yellow) to the output layer (blue), when the layers are flattened to 1D. The resulting *Toeplitz matrix* is well known in the context of unrolling convolutions as matrix-vector product (Chetlur *et al.*, 2014). Layers and kernels are color-coded by neuron and weight index, respectively. The two green boxes (labeled A, B) show two applications of the kernel at different locations of the feature map.

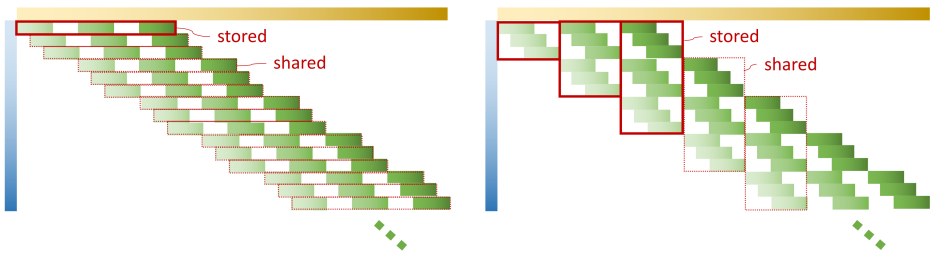


**Figure 3.6:** Nominal connection sharing in CNNs (left), and as implemented with NxTF (right).

With this perspective of the connectivity matrix at hand, we can compare weight sharing in a CNN on general-purpose hardware against weight sharing on Loihi

(Fig. 3.6). The left panel shows that, in a CNN, we would ideally store the kernel only once, and re-use it when iterating over the output neurons (rows). In Loihi however, this row-wise perspective has to be transposed to a column-wise processing: We have seen in Fig. 3.3 that a source neuron connects to its target neurons via a list of synapses that are represented by the nonzero entries in a column of Fig. 3.6. Such a column vector does not appear with the same regularity as a row vector. Thus, the current version of Loihi does not support weight sharing to the extent used in CNNs. However, the resource-sharing capabilities that are available allow for enough reuse to deploy large-scale DNNs. The basic principle is illustrated by the right panel of Fig. 3.6. A population of input neurons shares a group of synapses (solid red rectangle), which is stored on chip. Whenever one of these synapse groups appears at other spatial locations, the synapses can be reused by those other neuron populations as well (dotted red rectangles).

The degree to which we can exploit this connection sharing depends on the particular way that the layer is partitioned across neurocores. Connection sharing is factored into the partitioner's cost function and thus is subject to the optimization procedure discussed below.

### 3.2.3.2 *Axon Sharing*

Aside from neuron count and synaptic memory, another limited resource per neurocore is the number of core-to-core routing slots, the axons. Similar to sharing synapses, Loihi enables sharing axons for more efficient use of core resources. In Fig. 3.3 (right panel) we have shown how a population of source neurons (dashed blue box) can share the same axon. Likewise, we can define a population of input neurons in Fig. 3.6 (right panel), to contain all neurons of the input layer that utilize a particular synapse group (red box). This neuron population may use a single shared axon to send spikes to the output layer, where the spikes will be fanned out according to the synapse group. Without the ability to share axons, the layer would possibly have to be distributed across more cores to satisfy the axon constraint (Table 3.1).

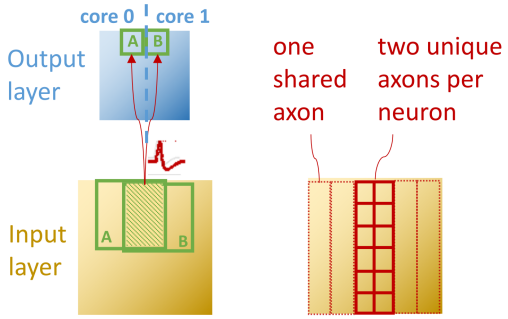


**Figure 3.7:** Unless targeting multiple cores, axons in NxTF may be shared. See text for details.

However, such axon sharing is possible only when spikes are being routed from one core to a single other core. If multiple cores are targeted, a discrete axon for each core is required. Fig. 3.7 illustrates this case in a convolution layer. The

subsequent application of a convolution kernel in two neighboring locations of the input layer defines a region of kernel overlap (shaded box). Neurons within this region send their spikes to two neighboring neurons $(A, B)$ in the output layer. If the layer happens to be partitioned such that neuron $A$ lives in core 0 but neuron $B$ in core 1, the input neurons within the shaded region each need two discrete axons to reach $A$ and $B$ on their separate cores. The input neurons outside the kernel overlap may share their axons as indicated on the right side of Fig. 3.7. As with connection sharing, the amount of shared axons is included in the cost function and optimization algorithm.

#### 3.2.3.3 *Cost Function*

As outlined at the beginning of Sec. 3.2.3, we search for a suitable partition by first proposing a number of partition candidates and then comparing their cost. The efficiency of a given layer partition is evaluated based on hard and soft constraints. Violation of a hard constraint (like exceeding the number of neurons per core) immediately invalidates a partition candidate. These limits are defined by available hardware resources on neurocores and are listed in Table 3.1. Soft constraints concern the flexible allocation of resources within the hard limits, and count towards the total cost of a partition within the optimization algorithm. For instance, distributing a network of 10 neurons across 10 cores ensures the hard constraints but is suboptimal if the neurons could fit on a single core.

The total cost of layer $L$ with respect to soft constraints can be written as

$$\mathcal{C}^L_{\text{soft}} = \alpha_0 \cdot N^L_{\text{cores}} + \alpha_1 \cdot \frac{N^L_{\text{syn}}}{N^{\max}_{\text{syn}}} + \alpha_2 \cdot \frac{N^L_{\text{axons}}}{N^{\max}_{\text{axons}}} + \alpha_3 \cdot N^L_{\text{offchip}}, \tag{3.1}$$

where the first term counts the number of cores, the second term the synaptic resources, the third the number of axons, and the last accounts for the cost of routing spikes from one chip to another (which costs twice the number of axons as for core-to-core connection). The synapse and axon cost terms are normalized per core for compatibility with the first term. The $\alpha_i$-coefficients allow custom weighting of different cost terms but are uniformly set to 1 in our experiments.

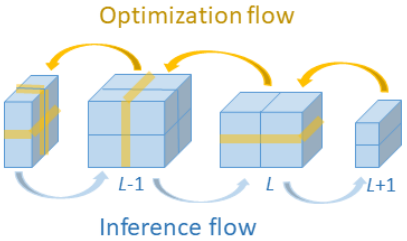


**Figure 3.8:** The way layer $L+1$ is partitioned (blue line denotes core border) effects a duplication of axons in layer $L$ for neurons connecting to multiple partitions / cores in layer $L+1$ (thick yellow line), thus determining the reverse optimization direction.

#### 3.2.3.4 *Optimization procedure*

We saw in Fig. 3.7 that a given partitioning of layer $L$ influences the resource allocation in the next lower ("pre-") layer $L-1$, namely the axon count in the depicted case. This circumstance implies that our optimization algorithm needs to traverse the network graph from top to bottom: A given layer can only be partitioned once we know the resource duplication imposed by its post-layer (Fig. 3.8).

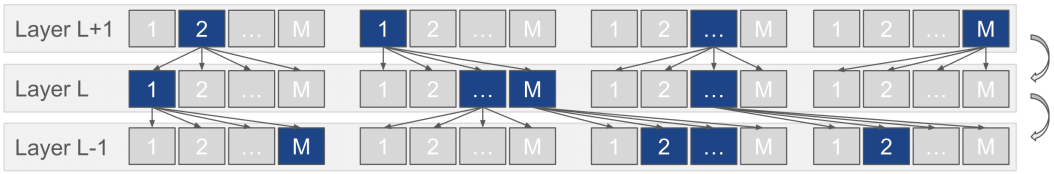


**Figure 3.9:** Each layer selects $M$ possible partitions (blue) among $M^2$ candidates according to cost function (3.1).

To optimize the distribution of a layer $L$, the partitioner proposes $M$ different *partition candidates* and computes their cost according to Eq. (3.1). But since layers cannot be optimized in isolation, the partitioner moves on to layer $L-1$ and proposes another $M$ partition candidates for each of the $M$ candidates of layer $L$. If we were to continue like this for a simultaneous optimization of the whole network, the combinatorial space would quickly become intractable because of the exponential growth in the number of partition candidates. Instead, we settle on a greedy optimization approach where the number of candidates per layer never exceeds $M^2$. After proposing $M^2$ candidates for layer $L-1$ ($M$ for each of the $M$ candidates of layer $L$), the partitioner selects the $M$ best ones according to the combined cost of layer $L-1$ and all its parent layers (index $\geq L$). Fig. 3.9 illustrates this selection process.

#### 3.2.3.5 *Synapse Compression*

To store synaptic memory on chip, Loihi supports several compression methods: `sparse`, `dense`, and `run-length`. Currently, this option is set by the user (or left at default). A future version of NxTF could determine the optimal compression scheme at compile-time.

#### 3.2.3.6 *Soft Reset*

A common way of resetting the membrane potential after a spike is to set it to zero. Rueckauer *et al.*, 2017 have shown that in a time-stepped simulation there is some information loss associated with this *hard reset*, which leads to an increased classification error. One alternative reset mechanism reduces the membrane potential by the threshold magnitude, which retains any excess charge and prevents information loss.

To support this *soft reset* mode in NxTF, we exploited Loihi's capacity for multi-compartment neurons. In *hard reset* mode, our cells are point neurons; in *soft reset*, they consist of two compartments, where one fulfills the role of the soma and

accumulates voltage to generate spikes. The second compartment is recurrently connected to the first and becomes active only when the soma crosses threshold and issues a spike. The second compartment then inhibits the soma via the recurrent connection with an amount of charge equal to the threshold.

The optional use of this *soft reset* implementation improves error rate and reduces runtime, at the cost of doubling the number of compartments allocated for the network.

#### 3.2.3.7 *Parameter Normalization*

When porting floating-point weights from an ANN to Loihi, two issues need to be considered. First, Loihi expects integers within a given bitwidth for weight, bias and threshold values. Second, previous work has shown that the limited dynamic range of SNNs can lead to saturating or vanishing firing rates unless the network parameters are properly rescaled (Diehl *et al.*, 2015b; Rueckauer *et al.*, 2017). To address these two points, we briefly summarize a normalization algorithm that ensures that parameters satisfy the hardware requirements, and that neuron activity spans the full dynamic range.

**INTEGER CONVERSION.** For the weights $W$ of each layer we apply the following steps:

1. Determine the maximum or some high percentile of the weight distribution: $\sigma = \max(W)$.
2. Normalize values in $W$ by $\sigma$.
3. Scale by the highest value allowed for weights, i. e. $\max(-W^-_{\text{lim}}, W^+_{\text{lim}})$.
4. Truncate values to the nearest integer.
5. Clip the resulting weight values to the allowed range $[W^-_{\text{lim}}, W^+_{\text{lim}}]$. Clipping is only necessary if $\sigma$ was computed based on a percentile rather than the max.

Biases can in principle be converted in the same way. A subtle difference arises from the fact that the allowed bit precision differs for weights and biases. If one wishes to fully exploit their individual bitwidth, determining the normalization and scaling factors is more involved; details can be found in the implementation.

**DYNAMIC RANGE.** To obtain the scaling factor for the dynamic range, we iterate over each layer in turn and estimate the expected distribution of net voltage change $\frac{\mathrm{d}u}{\mathrm{d}t} = Wx + b$ to neurons in the layer, using a subset of the training data. To estimate the expected input distribution without having to run the SNN, we build an SNN emulation from a modified copy of the ANN. In this model we perform the same changes to the weights as we would do to the SNN (i. e. integer quantization as outlined above), and mimick the SNN activation function by dividing by the threshold $\tau$, which converts the membrane potential change into a spikerate.

If the maximum (or some high percentile) $\lambda = \max(\frac{\mathrm{d}u}{\mathrm{d}t})$ of the input distribution lies near the voltage threshold $\tau$, then this layer is utilizing the full dynamic range,

and no change is required. Conversely, a supra-threshold value $\lambda > \tau$ indicates saturation, because neurons cannot fire more than one spike per time step. Small net inputs $\lambda < \tau$ predict that a large fraction of neurons will have low or possibly vanishing firing rates.

To prevent vanishing or saturating spike rates, the measured scaling factor $\lambda$ is used to scale the weights and biases as described in Rueckauer *et al.*, 2017. Then, the layer parameters are transformed to integers as described above. Finally, these values are decomposed into mantissa and exponent parts as expected by Loihi.

## 3.3 RESULTS

In this section we demonstrate the function of the NxTF compiler in two common use cases. 1) Frame-based ANNs are trained on standard image datasets, converted to SNNs and then evaluated on Loihi. 2) SNNs are trained on event-based datasets using SLAYER (Shrestha *et al.*, 2018), and are then deployed on Loihi. These two use cases demonstrate that the partitioning scheme described above applies to converted as well as SLAYER-trained models.

In addition to reporting the classification error for the tested models, we perform a profiling of power consumption and execution time, and compare against results published for other neuromorphic platforms.

### 3.3.1 SNN Performance Evaluation

Loihi supports measuring both execution time and energy consumption while running the SNN. In particular, we are interested in the energy and wall-clock time per inference sample. For a more fine-grained analysis, we can distinguish several phases during each algorithmic time step performed by Loihi. The first phase, *spiking phase*, takes care of transmitting spike messages among neurocores and updating the neuron's internal states. The next phases concern learning and are not executed here. The final phase is an optional *management phase* where the user can execute custom code such as needed for a global voltage reset at the end of a sample presentation, or for reading out the result at the output layer. Following the management phase, the embedded x86 processors either proceed to the next time step or instead may hand over control to the host CPU (counted as *host time*), e. g. for probing neuronal state variables. When comparing against results by other groups we use the total execution time per inference, i. e. the sum of all mentioned phases.

Power measurements likewise can be broken up into *static* and *dynamic* components, which cover leakage and switching of transistors, respectively. In addition, we can distinguish power consumption of the embedded processors from the neurocores. The dynamic power consumption of embedded processors is due to sustaining the clock and executing any user code. Neurocore dynamic power is proportional to the spike processing workload. Loihi's power measurement automatically excludes the static contribution of x86 processors that are not used by the model, as well as unused neurocores.

A useful metric to combine both execution time and power consumption is the Energy Delay Product (EDP) (Davies, 2019; Gaudet, 2014). It is computed from the product of energy and execution time (i. e. the "delay" to process a sample, for instance to reach a classification result). In our experiments, the energy per inference sample (or frame) is computed from the product of execution time per sample and the total dynamic and static power of neurocores and x86 processors. The motivation for using the EDP metric is that reporting power consumption alone can be misleading: Unless execution time is constrained by a given input frame rate in a real-time application, one could slow down the execution time (reduce clock rate or supply voltage) to achieve a desired level of power consumption. The EDP is a figure of merit that is minimized by the combination of power and execution time. It is similar to the power-latency-product used in Liu *et al.*, 2019, where "latency" is the equivalent to our "delay". The difference is that EDP contains an additional factor for the delay, thus weighing the execution time quadratically.

### 3.3.2 Models Trained with SLAYER on Event-Based Datasets

#### 3.3.2.1 *N-MNIST*

Our first use case for NxTF consists of deploying a model trained directly on spikes. We take as example a fully-connected network with a 1156-neuron input layer, a 512-neuron hidden layer, and a 10-neuron output layer. The model is trained on N-MNIST (Orchard *et al.*, 2015a) using the SLAYER method (Shrestha *et al.*, 2018). This spike-based training method uses a form of backpropagation that updates synaptic weights by taking into account the timing of spikes. The N-MNIST dataset is an event-based version of the classic MNIST handwritten digit dataset. The neuromorphic variant is obtained by recording the MNIST digits using a DVS, while performing saccades to elicit motion-induced events.

After constructing the model using the NxTF interface (c. f. Fig. 3.2), we transfer the publicly available weights from the original implementation[4] into our model, and invoke the NxTF compilation function. The network is mapped onto 30 neurocores (about 25% of a Loihi chip). Even though the number of neurons in this network is small, the all-to-all connectivity between layers leads to a large number of synapses in the hidden layer. Hence, the partitioner distributes this layer across 18 cores that are only partially occupied by neurons but whose synaptic memory is exploited to the limit. Similarly, the input layer is distributed across 11 cores, which do not exhaust the neuron capacity but rather the limit of available output axons (to accommodate the large fan-out to the hidden layer). The final layer (consisting of 10 neurons) fits on a single core, where it consumes less than 15% of available neuron-, synapse- and axon-resources.

As in the original SLAYER work, each of the 10000 N-MNIST test samples is run for 350 algorithmic time steps, resulting in a total classification error of 1.49% on Loihi. Table 3.2 compares the error rate against a model trained on spikes using spatio-temporal backpropagation (Patiño-Saucedo *et al.*, 2020) and deployed on the SpiNNaker platform (unfortunately, power and execution time measurements were

4 https://github.com/bamsumit/slayerPytorch, accessed Aug 2020

**Table 3.2:** Summary and comparison of results for N-MNIST. SpiNNaker results refer to Patiño-Saucedo *et al.*, 2020.

| Hardware | Num Neurons | Num Parameters | Error [%] | Time per Inference [ms] | Energy per Inference [mJ] | Energy Delay Product [$\mu$Js] |
|---|---|---|---|---|---|---|
| SpiNNaker | 810 | 792010 | 2.08 | - | - | - |
| Loihi | 522 | 596992 | 1.49 | 7.07 | 12.76 | 90.23 |

**Table 3.3:** Breakdown of power consumption [mW] for N-MNIST

| Hardware | | Static | Dynamic | Total |
|---|---|---|---|---|
| Loihi | x86 | 0.37 | 26.03 | 26.39 |
| | neurocores | 37.18 | 24.03 | 61.22 |
| | total | 1753.68 | 50.06 | 1803.74 |

not available). Table 3.3 lists the power consumption of the embedded x86 CPU and separates static and dynamic power. With this SLAYER-trained model, inference on N-MNIST can be done at 141 samples per second and with an energy consumption of 12.76 mJ per sample. A direct comparison against CPU and GPU performance is not possible here due to the event-based nature of the dataset, however, we will provide a comparison for the frame-based datasets in Sec. 3.3.3. Further results on the DVS-Gestures dataset (Amir *et al.*, 2017) are currently in preparation and will be added in the final version of this manuscript.

### 3.3.3 Converted Models on Frame-Based Datasets

Our second use case for the NxTF framework covers the conversion of ANNs to SNNs. For convenience we developed an interface between NxTF to a common conversion software, the *SNN toolbox* (Rueckauer *et al.*, 2017). This open-source framework automates most stages in the conversion and deployment pipeline. A user provides the pre-trained ANN in one of the supported DL libraries (Tensorflow / Keras, Pytorch, Caffe, Lasagne). The toolbox parses the provided network and constructs an equivalent model using the NxTF layer classes. If desired, the toolbox performs a parameter normalization step as outlined in Sec. 3.2.3.7. Subsequently, the network is compiled and run on Loihi. The toolbox includes methods for performance benchmarking and visualization of network activity.

We used the toolbox to convert and run a CNN trained on MNIST. Further results on CIFAR-10 using a MobileNet architecture (Howard *et al.*, 2017) are in preparation and will be included in the final version of this manuscript.

#### 3.3.3.1 *MNIST*

For MNIST we trained a 4-layer CNN with Keras (Chollet, 2015) to achieve an error rate of 0.74%. After conversion to an SNN using the rate-based encoding method by

Rueckauer *et al.*, 2017, the SNN was mapped to 14 neurocores on Loihi. Even though the model contains about 8× more neurons than the N-MNIST model in Sec. 3.3.2.1, the core count is halved because the convolutional architecture requires about 88× fewer parameters than the fully-connected one. Thus, the utilization of resources on neurocores is dominated by the neuron count in case of the CNN, rather than synapse / axon count as in the fully-connected network. The reduced core count also highlights the benefit of the basic connection sharing capabilities of Loihi as described in Sec. 3.2.3. If no connections could be shared, the CNN would require 341472 discrete rather than 6746 shared weights, and the core count would likely approach that of the fully-connected model.

**Table 3.4:** Summary and comparison of results for MNIST. Minitaur refers to FPGA results from Kiselev *et al.*, 2016. SpiNNaker refers to results from Stromatias *et al.*, 2015. TrueNorth refers to Esser *et al.*, 2015. BrainScales refers to Schmitt *et al.*, 2017, who used 5 out of 10 classes and a subset of 1200 samples with resolution reduced to $10 \times 10$. SpiNNaker* refers to Patiño-Saucedo *et al.*, 2020. Loihi* refers to Massa *et al.*, 2020.

| Hardware | Num Neurons | Num Parameters | Error [%] | Time per Inference [ms] | Energy per Inference [mJ] | Energy Delay Product [$\mu$Js] |
|---|---|---|---|---|---|---|
| Minitaur | 1010 | 648010 | 8 | 136 | - | - |
| SpiNNaker | 1010 | 648010 | 4.99 | 11 | 3.3 | 36.26 |
| TrueNorth | 3840 | - | 0.60 | 1 | 0.11 | 0.11 |
| BrainScales | 55 | 2155 | 5 | - | - | - |
| SpiNNaker* | 7614 | 1236520 | 1.80 | - | - | - |
| Loihi* | 58402 | 60068 | 1.30 | 8 | - | - |
| Loihi (ours) | 4314 | 6746 | 0.79 | 6.65 | 12.06 | 80.22 |

**Table 3.5:** Breakdown of power consumption [mW] for MNIST

| Hardware | | Static | Dynamic | Total |
|---|---|---|---|---|
| Loihi | x86 | 0.36 | 22.29 | 22.65 |
| | neurocores | 16.96 | 59.24 | 76.20 |
| | total | 1730.88 | 81.53 | 1812.41 |

After mapping the model onto Loihi, we ran it for 100 algorithmic time steps per sample, reaching 0.79% error rate. Inference was possible at 150 frames per second (fps), with an energy consumption of 12 mJ per sample. Table 3.4 compares this result against related published work; Tab. 3.5 breaks down the power consumption of embedded CPUs and neurocores.

A useful property of SNNs is their ability to trade off accuracy against computational cost. As the classification result is obtained by accumulating evidence (spikes) at the output layer across a certain time period, one may shorten the run time of the network to reduce the inference delay and energy cost at the potential risk of

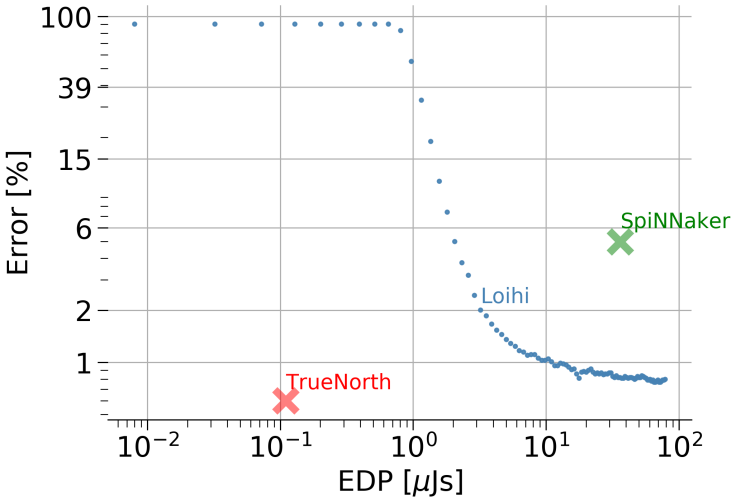


**Figure 3.10:** Trading off classification error against EDP.

misclassifying some samples. Fig. 3.10 displays the resulting trade-off curve for the MNIST model tested here.

## 3.4 DISCUSSION

In this work we introduced NxTF, a software package to deploy DNNs on the neuromorphic chip Loihi. In this task we were guided by two objectives: User-friendly and straight-forward access to SNN hardware, and optimal usage of the limited resources on chip. The first objective was achieved by building our user-interface on top of Keras / Tensorflow. The second objective was approached using a greedy layer-wise optimization algorithm that minimizes a cost function counting the neuron-, axon-, and synapse-resources on Loihi cores. Further, we exploited Loihi's support for connection sharing to minimize duplication of synaptic weights in convolutional architectures.

As a proof of concept, the NxTF framework was applied in two typical use-cases: Running models obtained via direct spike-based training, and running converted models on frame-based data. For the first use-case we considered the neuromorphic dataset N-MNIST. A model trained with SLAYER directly on the input spikes was mapped to Loihi and achieved 1.49% classification error at 141 samples per second and 12.76 mJ per sample. In the second use case we trained a CNN on MNIST frames, converted it to SNN, and ran it on Loihi using the rate-coded frames. The model achieved an error rate of 0.79% at 150 fps and 12.06 mJ per sample.

For the MNIST dataset, results on two other neuromorphic platforms are available. On one hand, an experiment on SpiNNaker (Patiño-Saucedo *et al.*, 2020) demonstrated an improved energy efficiency compared to our result at a slightly reduced inference speed and significantly higher classification error (c. f. Table 3.4). According to the error-vs-performance trade-off in Fig. 3.10, our model reaches the same error at a 18× smaller EDP. On the other hand, our model has an 714× increased EDP compared to TrueNorth (Esser *et al.*, 2015), which is mainly thanks to the high

energy efficiency of TrueNorth (0.11 mJ per sample). This remarkable difference can be attributed to the fact that models on TrueNorth are state-less and binarized, and neuron activations are encoded by single spikes rather than firing rates.

Preliminary results on the CIFAR-10 dataset show an error rate of 8.77%, which improves previous state-of-the-art (17.5% on TrueNorth (Esser *et al.*, 2016)) of this dataset on neuromorphic hardware by 2×. A performance analysis and comparison against CPU and GPU is in progress and will be included in the final version of the manuscript.

The present work highlights at least two areas of improvement in the research of SNNs: First, the relatively high EDP obtained in our experiments underlines the need for spike codes that are more efficient than the rate code employed here. This need has long been recognized, and a variety of alternatives have been proposed (Mostafa, 2018; Rathi *et al.*, 2020; Rueckauer *et al.*, 2018; Sorbaro *et al.*, 2020; Yousefzadeh *et al.*, 2019; Yu *et al.*, 2020; Zambrano *et al.*, 2019). However - and this is the second area of improvement - performance results in the form of energy and execution time per sample are scarce in the literature, even for well-explored datasets like MNIST. We hope that the SNN compilation framework developed here will enable a more rapid and straightforward evaluation of novel SNN models on neuromorphic hardware.

# 4 EVENT-BASED SENSING WITH THE DYNAMIC VISION SENSOR

In this Chapter we highlight some of the benefits and challenges of combining event-based *processing* (Chapters 2, 3) with event-based *sensing*. We begin with an example of data compression (Sec. 4.1.1) and optical flow (Sec. 4.1.2) before moving on to Spiking Neural Networks in Sec. 4.2. The chapter concludes with Sec. 4.3 by evaluating the potential of using a DVS to drive mouse RGCs in vitro.[1].

## 4.1 MOTIVATING EXAMPLES

### 4.1.1 Data Compression

Fig. 4.1 shows an example of how using the DVS leads to reduced recording data from an overhead view of a mouse in a cage. A researcher might be interested in studying the whisker motions of the mouse. These whisking motions require a high sample rate of 1 kHz which is easily supplied by the DVS pixel bandwidth. We observe from the 3.5 day recording that there are long periods of near silence from the sensor followed by brief bursts of high activity as the mouse explores the cage. The event rate probability follows a distribution where the log probability decreases approximately linearly and inversely with the event rate. It means that the output event rate is dominated by mostly low rates, with a tail of high rates encoding the rapid motions. A 1 kfps image sensor would produce about 300 M frames or 5 TB of data from the equivalent resolution $128 \times 128$ pixel sensor. The DVS recorded 18.5 M events or 74 MB of 4-byte event data, which is a factor of $67,000\times$ less data. Of course a standard camera output could be greatly compressed, but its output would require at least initial processing at this high frame rate, and this processing (and its latency) is avoided by using a DVS camera.

### 4.1.2 Tradeoff between Accuracy and Computational Cost in Optical Flow

To better understand the accuracy and computational cost tradeoff using event sensors, we look at this tradeoff when computing optical flow directly from the events; and from frames based on events. For this experiment, we use constant time frames so we can choose arbitrary frame rates.

The experimental setup is as follows: A white disk with a black bar near its outer edge is mounted on a motor with controllable speed. A DAVIS camera records the disk while we ramp the rotational speed up and down. The rotational frequency of the disk increases to a maximum of about 12 Hz within 3 s, holds that level for

1 The material in this Section is based on two sections of Liu *et al.*, 2019, where I am second author and main contributor to the material listed here.

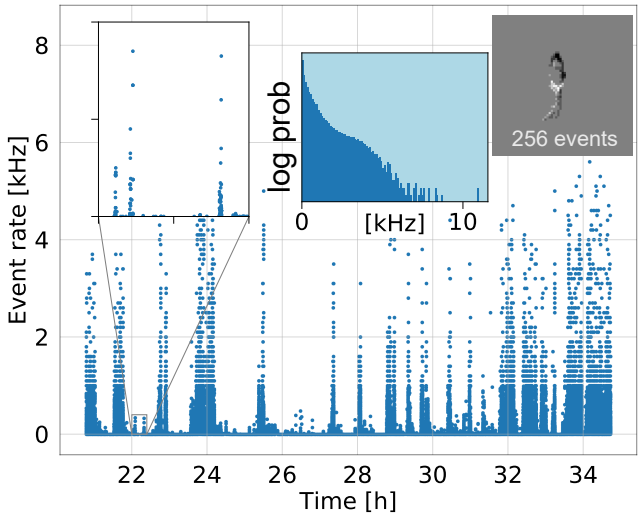


**Figure 4.1:** Event rate of a DVS recording of overhead view of mouse (right inset) over 14 hours. Peaks correspond to high levels of motion and therefore potential regions of interest. The middle inset shows a histogram of the event rates for the complete 3.5 day recording, using a log scale for the vertical axis. (Data from unpublished collaboration with Irene Tobler, University of Zurich.)

another 5 s, and finally decreases continuously to 5 Hz over a period of 7 s (Fig. 4.2b). The leading edge of the dot on the disk generates a stream of OFF events from the DVS. We filter out the ON events from the trailing edge, which are not needed for this task. Background events are also discarded using a spatio-temporal correlation criterion (Delbruck, 2008). The median event rate is then 120 kHz. Fig. 4.2a illustrates the experimental data.

For event-based flow computation we used the Savitzky-Golay variant of the "LocalPlanes" algorithm described in Rueckauer *et al.*, 2016a. The algorithm maintains a "time surface" (Benosman *et al.*, 2014) corresponding to the most recent event timestamps for all pixels, and attempts to fit a 2D plane in the $3 \times 3$ neighborhood around each incoming event pixel address. The normal flow of the edge causing the event is then obtained from the slope of the fitted plane.

We estimate the rotational frequency from the flow vectors by projecting them onto the tangents along the circular trajectory. The event data curve in Fig. 4.2b shows the resulting rotation frequency over the duration of the recording. (Of course, knowing that we are observing a rotating dot, it is clear that the rotation frequency of the disk could have been obtained from methods other than flow, but we use this metric as a proxy for the accuracy of the flow computation.)

For the frame-based flow computation, we use the standard `opencv` implementation, which consists of a key-point tracker combined with a Lucas-Kanade (LK) sparse flow algorithm. Parameters of the LK algorithms are optimized to obtain the highest accuracy. We generated frames for LK from the continuous DVS event stream at frame-rates varying from 50 to 300 Hz. Each frame is a 2D histogram of the DVS events collected during the frame interval (c. f. Sec. 4.2.2.2).

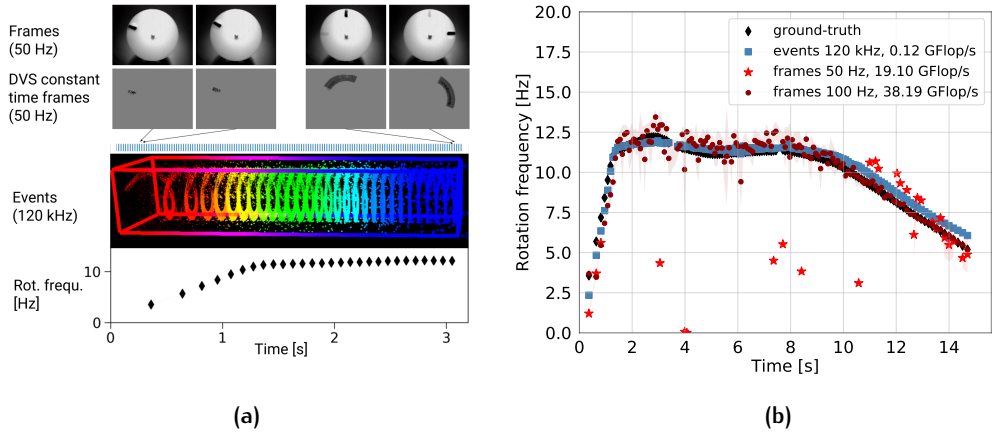


**Figure 4.2:** (a) Illustration of the spinning disk data. (b) Rotational frequency measurements from optical flow over the duration of a DVS recording of a spinning disk, estimated using an event-based optical flow algorithm (blue), and a frame-based Lucas-Kanade method (red). Ground truth (black diamonds) for the motion of the disk is obtained by measuring the timestamps of events produced when the dash passes a $1 \times 30$ pixel window, and then k-means clustering the times of events obtained in this window. The timestamps of a single pixel initialize the k-means algorithm.

Fig. 4.2b shows that since the event-stream is near-continuous (with a time resolution of 1 $\mu$s), the event-based flow algorithm does not suffer from aliasing and accurately estimates the motion at all speeds (blue squares). The frame-based flow algorithm is able to accurately estimate the optical flow over the whole duration of the recording, but only if the frame rate is 100 Hz or higher. At the lower frame rate of 50 Hz, fast motion (greater than 10 cycles/s) results in large displacements of the rotating bar across consecutive frames, and consequently high errors of the LK algorithm.

This tradeoff between frame rate (a proxy for computational cost) and accuracy is demonstrated in Fig. 4.3, which shows how the Mean Square Error (MSE) of the estimated rotational frequency decreases with higher sampling rates. The frame-based sparse LK algorithm requires 382 M Flop per frame, while the event-based flow computation costs 980 Flop per event (Rueckauer *et al.*, 2016a) and therefore on average 0.12 GFlop/s for this recording. To achieve the same level of accuracy as the event-based method, the frame-based algorithm requires a frame-rate of at least 100 Hz, costing about 38 GFlop/s, which is a factor of about 320× more.

This experiment illustrates the advantage of data-driven dynamic sampling which in this case, enables the event-based flow algorithm to accurately solve the task at high object velocities without special knowledge of the rotating dot model that could accommodate large displacements for fast motion.

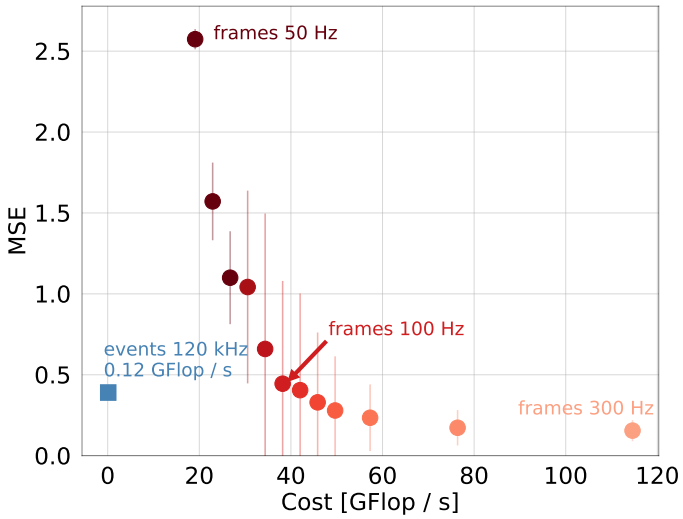


**Figure 4.3:** MSE of the extracted rotational frequency against the computational cost with different event frame rates (circular disks color-coded in red). The blue square marks the accuracy and cost of the event-based flow algorithm.

## 4.2 SPIKING NEURAL NETWORKS WITH ASYNCHRONOUS INPUT

Mobile and embedded applications require neural networks-based pattern recognition systems to perform well under a tight computational budget. In contrast to commonly used synchronous, frame-based vision systems and CNNs, asynchronous, spiking neural networks driven by event-based visual input respond with low latency to sparse, salient features in the input, leading to high efficiency at run-time. The discrete nature of the event-based data streams makes direct training of asynchronous neural networks challenging. This work studies asynchronous spiking neural networks, obtained by conversion from a conventional CNN trained on frame-based data. As an example, we consider a CNN trained to steer a robot to follow a moving target. We identify possible pitfalls of the conversion and demonstrate how the proposed solutions bring the classification accuracy of the asynchronous network to only 3% below the performance of the original synchronous CNN, while requiring 12x fewer computations. While being applied to a simple task, this work is an important step towards low-power, fast, and embedded vision solutions for robotic applications.[2]

### 4.2.1 Introduction

Deep CNNs offer practical solutions for pattern recognition and have radically changed the field of image recognition. In the field of robotics, however, where real-time processing and low power budget are crucial, CNN-based image-processing algorithms face a fundamental latency-power trade-off, where low latency can only be achieved by dramatically increasing power consumption.

Evolution of embedded systems led to development of event-based vision, which has enabled improved performance of vision systems for fast and agile robots (Falanga *et al.*, 2017; Mueggler *et al.*, 2017). Event-driven, biologically inspired vision sensors such as DVS (Lichtsteiner *et al.*, 2008), ATIS (Posch *et al.*, 2011), and Dynamic And Active Pixel Sensor (DAVIS) (Brandli *et al.*, 2014) enable fast and low-power processing of visual information. Instead of capturing static images of the scene, these sensors record pixel brightness change events with high temporal precision. Events are only triggered if a significant change occurs in the observed scene, allowing lower latency and lower required bandwidth compared to frame-based sensors. However, since the data produced by an event sensor is a sequence of events, conventional frame-based computer-vision algorithms (Bradski, 2000) or DNN-based pattern recognition can not be applied directly.

One obvious way to process event-based data with conventional DNNs is to create frames by accumulating events over fixed time intervals, or accumulating a constant number of events per frame. It has been shown in several robotic applications that by following this approach, conventional CNNs can be applied for feature extraction and object classification (Amir *et al.*, 2017; Lungu *et al.*, 2017; Moeys *et al.*, 2016). Although using constant event count frames addresses the latency-power tradeoff

2 The first part of Sec. 4.2 is based on Rueckauer *et al.*, 2019a, the second part presents unpublished work on the Roshambo hand gesture recognition task.

by using data driven computation, it ignores key advantages of event based sensors, in particular their sparse data and the high temporal precision.

This paper explores the use of asynchronous neural network architectures for processing the event-based vision data. In contrast to the synchronous, frame-based mode of operation of conventional CNNs, asynchronous SNN architectures represent hidden layer activations in form of discrete events – spikes – that are propagated through the network asynchronously, so that neurons are only activated when they receive events (Martin *et al.*, 2015). Theory has shown that SNNs are at least as computationally powerful as conventional neuronal models being used in deep-learning (Maass *et al.*, 2004). It has also been shown that by the use of dedicated event-based hardware, power consumption and latency can be reduced by several orders of magnitude (Furber *et al.*, 2013; Indiveri *et al.*, 2015; Merolla *et al.*, 2014). IBM's TrueNorth processor (Merolla *et al.*, 2014) consumes about 1000 times less energy than conventional synchronous architectures. Thus, just as hardware acceleration through GPUs has played a fundamental role in the advancements of deep-learning, there is increasing availability of neuromorphic SNN accelerators that enable efficient computation of event-based SNN training and inference (Lee *et al.*, 2016), potentially running on a fraction of the energy budget compared to conventional CNNs running on GPUs (Furber *et al.*, 2013; Indiveri *et al.*, 2009; Merolla *et al.*, 2014; Qiao *et al.*, 2015).

There are two ways to obtain an SNN for solving a pattern recognition task. First, recent work has explored direct training in the spiking domain using backpropagation inspired techniques for training multi-layer SNN architectures (Lee *et al.*, 2016; Neftci *et al.*, 2017). Training SNNs is difficult as due to their non-differentiable nature and gradient-descent based methods can not be applied directly. Furthermore, backpropagation rules typically used in deep learning rely on the availability of network-wide information stored with high-precision memory, and on precise operations that are difficult to realize in event-based hardware (Neftci *et al.*, 2017).

Second, SNNs can be constructed by converting conventionally trained ANN (Diehl *et al.*, 2016b; Rueckauer *et al.*, 2017). In terms of accuracy, Stromatias *et al.*, 2017 reports that while these neural networks seem to work well using synthetic input spike trains generated artificially from frame images (e. g. from the MNIST database), where the gray level of an image pixel is transformed into a stream of spikes, doing inference using this SNN with data from an event-based vision sensor may lead to significant loss in accuracy. Increasing our understanding of SNN processing is needed to close the accuracy gap between the frame-based and event-based pattern recognition.

In this Section, we apply the second method and analyze object recognition using analog and spiking convolutional neural networks in the context of a robotics predator/prey navigation scenario. The dataset from Moeys *et al.*, 2016 is used to train and evaluate several neural network architectures, where the purpose of the trained networks is to steer a predator robot in the direction of a prey robot. In particular, we compare the conventional CNN architecture proposed in Moeys *et al.*, 2016 with its event-based SNN counterpart, where accuracy is evaluated using both synthetic and sensor-driven input spike trains. We perform a thorough analysis of accuracy losses that occur in the ANN to SNN conversion and offer solutions to

reduce these losses. We show that a CNN trained on constant event count frames can be run efficiently on the asynchronous sensor events at inference, using up to 12x fewer computations than when using frames. Finally, we identify the causes for classification accuracy loss that occurs when switching from a synchronous training mode to an asynchronous inference mode, and evaluate solutions that minimize this loss.

These are crucial steps on the way to low-power, fast, embedded solutions, which will enable the application of deep neural networks on robotic platforms in real time and with a limited power budget.

### 4.2.2 Methods

This subsection describes the pipeline of the present work: Starting with an event-based data set (Sec. 4.2.2.1), we first synthesized frames (Sec. 4.2.2.2) to train a conventional ANN (Sec. 4.2.2.3). The resulting frame-based model was then converted to an SNN (Sec. 4.2.2.4) and tested on the original event-based data (Sec. 4.2.2.5).

#### 4.2.2.1 *The Dataset*

The dataset from Moeys *et al.*, 2016 consists of twenty recordings with a total duration of 1.25 hours from a DAVIS (Brandli *et al.*, 2014). The DAVIS camera was mounted on the predator robot and recorded different scenes in which the predator robot, driven by a human or by the CNN, followed the prey robot. The recordings contain both conventional image frames (Active Pixel Sensor (APS)) as well as event-based data (DVS). The APS frames were not used in this work. The DVS sensor data was output in Address Event DATa (AEDAT) 2.0 file format (INIlabs, 2010): each sensor event contained a timestamp, the pixel address, and a polarity value (ON/OFF), indicating an increase or decrease in pixel brightness.

The ground truth labels of the recordings encoded the position and bounding box of the prey robot. From these labels, we produced a target ground truth of four classes, marking in which third of the visual field the prey robot is located (classes 1-3) or if it is not visible (class 4), leading to a four-class-classification problem (left, center, right, invisible). The DAVIS camera records with a resolution of 240x180 pixels. We subsampled the event-addresses to 36x36 arrays. Moeys *et al.*, 2016 found this to be the minimum size for which the robot can still be recognized by human eye. The data set consisted of roughly 200k images generated by binning DVS events to 5000-event frames, as described below.

#### 4.2.2.2 *Generating Frames from Event-Based Data for ANN Training*

Training of the ANN was conducted on frames synthesized from DVS events, but testing of the converted SNN was performed on the original DVS stream. Thus, any transformation of the data during frame synthesis should either be applicable to the underlying DVS event streams as well, or else distort them as little as possible. This subsection describes each step of a frame generation method that best preserves the classification performance when using asynchronous DVS data at test time.

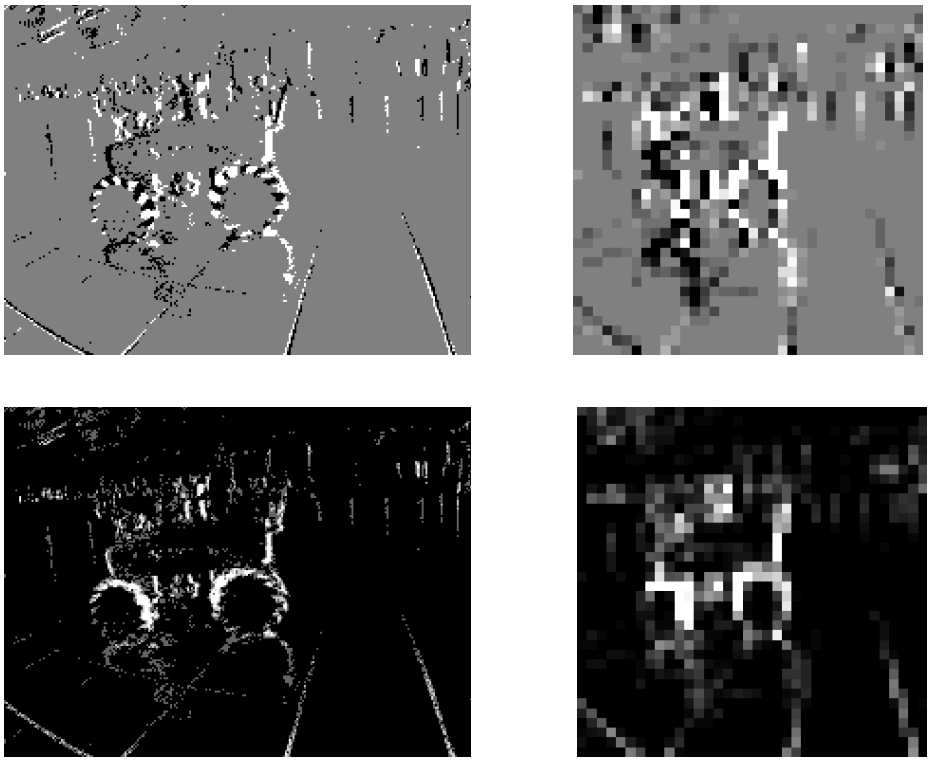

**Figure 4.4:** Synthesized DVS frames (see Sec. 4.2.2.2 for details). Top row: Original method from Moeys *et al.*, 2016. Bottom row: Our method. Left column: Full resolution (240x180); right column: Subsampled to 36x36.

**CHOOSING THE BINNING WINDOW** Frames can be synthesized from DVS data either by accumulating a variable number of events during a fixed time window, or by accumulating a fixed number of events during a variable time window. We follow Moeys *et al.*, 2016 and use the latter approach, with a constant number of 5000 events per frame. This way, the frame rate is proportional to the rate of change of the scene, and each frame is more likely to be informative. Frame synthesis with a fixed time window can lead to overly sparse and noisy frames during time intervals with few changes in the recorded scene, and blurred frames when the robots are moving quickly.

**HANDLING POLARITY** Another design choice concerns the polarity of the DVS events. One can integrate ON and OFF events by representing them as +1 or -1, respectively. In this case, events of opposite polarity cancel each other. The original work Moeys *et al.*, 2016 uses this method by initializing the frames with 0.5 pixel intensity and in-/decrementing this value by $\pm 0.005$ per ON/OFF event. This approach is not feasible in our setup, because it assigns a nonzero intensity to pixels where the DVS records no events. Instead, we start with an all-zero frame and apply a rectified event count that discards polarity while binning the events. In preliminary experiments we found the polarity information not to be relevant for learning this task.

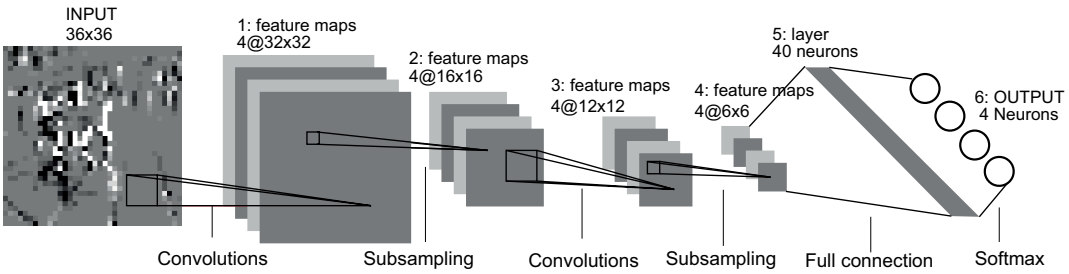


**Figure 4.5:** Architecture for predator-prey task. (Graphic adapted from LeCun *et al.*, 1998)

**INPUT NORMALIZATION** Outliers in the distribution of event counts were removed by clipping values greater than three times the standard deviation (3-sigma normalization).

Though the network was trained on frames, we aimed to use the original DVS events during inference. To maintain high classification accuracy when switching from frame to event input, any transformation of the frame data, performed during training, should be applied to the DVS data as well. 3-sigma normalization on DVS event streams is possible by temporally binning 5000 events into a frame as during training, and applying 3-sigma normalization on this frame of integer event-counts to identify outlier events, which are then removed from the DVS event stream fed into the SNN.

**INPUT SCALING** After accumulation and outlier removal, the frames are scaled to $[0, 1]$ real values, which is the last stage of synthesizing frames for training the ANN. During testing of the SNN, discrete events are streamed into the network, making scaling inapplicable.

A subtle difference in the training-frames arises from the order in which scaling and 3-sigma normalization are applied. In Moeys *et al.*, 2016, the frame of integer event-counts is scaled to $[0, 1]$ before 3-sigma normalization. The resulting frames consist of real values. If instead 3-sigma normalization is applied first (by removing discrete events from the frame of integer event-counts), the subsequent scaling will result in frames that consist of rational numbers only. To avoid the discrepancy of training on real values and testing on integers, we applied 3-sigma normalization *before* scaling. This seemingly small difference turns out to be crucial: With the opposite ordering, the classification accuracy of the converted SNN drops by 30%.

Fig. 4.4 shows an example of the frames synthesized as described above. In Table 4.1 we summarize how our approach differs from the original work (Moeys *et al.*, 2016).

#### 4.2.2.3 *Training the Frame-Based ANN*

Fig. 4.5 shows the model architecture that Moeys *et al.*, 2016 developed to solve the predator-prey task. It consists of a small CNN with two convolution layers with 4 feature maps each and a kernel size of 5x5 pixels. Each convolution layer is followed by a max pooling layer. A fully-connected layer of 40 neurons connects the last pooling layer with the 4 classifier output units. The network contains a total of 5884

**Table 4.1:** Comparison of methodology.

| | Moeys *et al.*, 2016 | This work |
|---|---|---|
| APS frames used | Yes | No |
| Biases | Yes | Yes, with L2-regularizer |
| Event polarity used | Yes | Yes, rectified |
| Frames initialized at | 0.5 | 0, for comput. sparsity |
| Subsampling (cf. 4.2.3.2) | sum | max, to remove clusters |
| Scaling frames to $[0, 1]$ | Yes | Yes (N/A in SNN) |
| 3-sigma normalization | Yes | Yes, before scaling |

neurons and 6472 parameters. Moeys *et al.*, 2016 showed that this tiny CNN achieves higher classification accuracy than humans observing the same images.

We implemented the network in the Keras framework (Chollet, 2015), and trained for 30 epochs using mini-batches of size 32 and the ADAM optimizer.

#### 4.2.2.4 *Converting the ANN to an Event-Driven SNN*

As described in detail in Sec. 2.1, the central idea of ANN-SNN conversion is that the time-averaged firing-rates of the resulting spiking architecture correspond to the analog activations in the original ANN. This mapping can be achieved by replacing the neurons in the ANN with non-leaky integrate-and-fire neurons (Burkitt, 2006). The trained parameters remain the same, up to a layer-wise rescaling that reduces the problem of limited dynamic range of spiking neurons (Diehl *et al.*, 2015b). Rueckauer *et al.*, 2017 proposed implementations in the spike domain of modules commonly used in ANNs, like max-pooling and softmax layers. We apply their open-source conversion framework (Rueckauer, 2017) to transfer the predator-prey ANN to the event-based domain.

#### 4.2.2.5 *Data Used for Testing the Converted SNN*

Due to the lack of truly event-based datasets acquired with neuromorphic vision sensors, in recent work the spiketrains were often generated synthetically from frame-based image datasets. The most common method is to use poisson spike generators driving their firing-rate with the intensity of the corresponding input pixels. However, the stochastic nature of the generated Poisson spike trains introduces noise into the network, without having any notable benefits. A simple alternative is to use analog input values in the very first hidden layer, and to compute with spikes from there on (Rueckauer *et al.*, 2017; Zambrano *et al.*, 2016). The image pixel values are interpreted as currents flowing into the neurons of the first hidden layer, where they are integrated into membrane potentials, thus deterministically producing regular spikes at a rate proportional to the pixel value. Recent work (Stromatias *et al.*, 2017) has reported that while converted SNNs seem to work well on synthetic input data, using real event-based data as input can lead to a significant drop in classification accuracy.

In this work, we perform simulations with *poisson* and *analog* inputs from synthetically generated frames (Sec. 4.2.2.2), and we compare these to directly applying the original DVS events from the predator/prey dataset as input spikes.

#### 4.2.2.6 *Simulation of the Converted SNN*

We make use of the SNN toolbox (Rueckauer, 2017) to run the converted SNN on the three input types described above. The SNN toolbox provides a simulator for spiking networks that is built on the Keras framework. The spiking network consisting of non-leaky integrate-and-fire neurons is processed in a time-stepped manner with a step size equivalent to the time resolution of the DVS event stream (1 microsecond).

The DVS data set used for testing the converted SNN is stored as a collection of *.aedat* files, where each file contains a DVS clip of several seconds. Previously, the toolbox accepted frame-like input. To be able to use asynchronous data, we extended the toolbox by a DataGenerator module that iteratively reads in an .aedat file and processes the event sequence with subsampling and outlier removal as described in Sec. 4.2.2.2. The network outputs a classification guess at each time step, but we define the period of time needed to process 5000 events as "one sample", and take as final classification output of the network for one particular sample the class corresponding to the neuron that fired the most spikes while processing the sample. When all events in an .aedat file are processed, the DataGenerator loads the next sequence of events from the aedat-directory; this procedure is repeated until all events are processed.

### 4.2.3 Results

#### 4.2.3.1 *ANN Accuracy*

First, we reproduced the results of Moeys *et al.*, 2016 by training the frame-based CNN architecture in Fig. 4.5 on frames generated from the DVS events as outlined in Sec. 4.2.2.2. The original work used a combined dataset of APS frames and synthesized DVS frames. We found that the same classification performance can be achieved using DVS frames alone, which is preferable in the present setup, because only DVS events will be used during inference.

#### 4.2.3.2 *SNN Accuracy*

After transferring the ANN to an SNN as described in Sec. 4.2.2.4, the SNN was tested on the three input types listed in Sec. 4.2.2.5, namely *analog* (frame-based), *Poisson*, and *DVS* input. Both analog and Poisson input resulted in SNN accuracies close to the original ANN accuracy (see Table 4.2). However, in our initial experiments, the SNN accuracy dropped to chance level when using the original DVS input spike trains. We discuss the reasons for this reduction of accuracy, and propose and evaluate solutions to restore accuracy in the remainder of this subsection.

**IMBALANCE BETWEEN NETWORK BIASES AND DVS RATES** In Rueckauer, 2017, the bias values of neurons in the ANN are converted into constant input currents

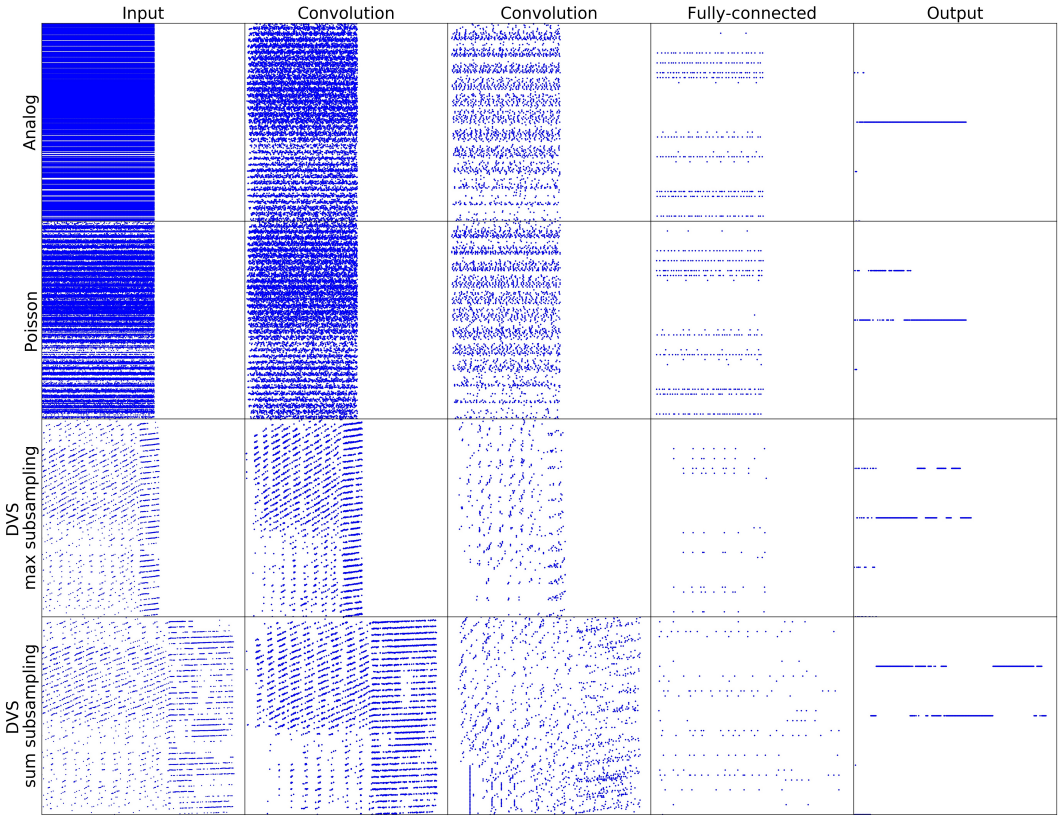


**Figure 4.6:** Spike trains generated by simulating the SNN on a single test sample corresponding to 5000 DVS events. X-axis: time (450 steps of simulation); y-axis: neuron index. See Sec. 4.2.2.5 for a description of the input types, and Sec. 4.2.2.2 for details on the frame generation. The Figure is discussed at the end of Sec. 4.2.3.2.

that flow into SNN neurons over the course of the simulation. If a bias value is large, this bias current can outweigh the spike-driven input to a neuron and dominate that neuron's output firing dynamics. This effect is likely to occur in neurons receiving DVS input spike trains, whose rates may vary considerably over the duration of a single 5000-event sequence (see Sec. 4.2.3.2).

To prevent dominating biases, we trained the ANN with L2-regularization on the network weights and biases. L2-regularization adds to the training cost function a term which is proportional to the squared parameter values, thereby inducing the network to keep parameter values small. Training the ANN without regularization led to several bias values that were close to or above the threshold of the SNN neurons, thereby dominating their firing dynamics. The classification accuracy of the converted L2-regularized SNN increased by 43% as compared to the SNN without regularization.

Training the ANN altogether without biases (as done in Diehl *et al.*, 2016b) may be another straight-forward solution. We trained an ANN without biases, which achieved 0.6% lower accuracy than the L2-regularized ANN containing biases. The converted SNN without biases scored better than the non-regularized SNN with

biases, but 0.81% worse than the L2-regularized SNN with biases. Thus, we favor the regularized model with biases.

**SUBSAMPLING INDUCES TEMPORAL CLUSTERS** Aside from dominating biases, another reason for the drop in classification performance when using DVS input was the subsampling mechanism. Pixel addresses in the original 240x180 image space are subsampled to 36x36 by integer division. If a subsampled patch contains several simultaneous events, they will all be mapped onto a single pixel address, thereby transforming a spatial-temporal cluster into a temporal cluster. A neuron in the SNN then receives the spikes contained in such a "burst" at immediately-subsequent time steps during the simulation. These spike bursts are in strong contrast to the way the network was trained, namely on analog frames, where such temporal structures are not present.

To see why temporally structured spike trains may produce a different outcome than homogeneously distributed spike trains, consider a neuron receiving a fixed number of input spikes from two sources, one inhibitory and the other one excitatory. If both sources fire at a regular rate, their contributions cancel each other and the neuron will not be active. If instead the spikes of the excitatory source are clustered into an early spike burst, the neuron will be strongly activated, even though the total number of spikes from each source over a given time period has not changed.

To prevent formation of detrimental temporal clusters during subsampling, we keep only one of the subsampled events in a patch. Here we term this method *max subsampling*, and the method that accumulates all events in a patch *sum subsampling*. The classification accuracy of the ANN trained on the max- and sum-subsampled data is 88.25% and 88.04%, respectively. The accuracy of the converted SNN is 85.19% and 78.24%, respectively, which shows the importance of removing spike bursts due to subsampling.

**NON-UNIFORM DVS SPIKETRAINS** With regularized biases and max-subsampling, the classification accuracy of the converted SNN using DVS input improves from chance level to within 3% of the original ANN. We were not able to close this gap completely, and believe the underlying cause to be inhomogeneities in the DVS spike trains.

By viewing the DVS recordings as well as by studying the raster plots in Fig. 4.6, one can observe phases of increased global activity within the time window that corresponds to 5000 events. These bursts of global activity are likely the result of electrical coupling between frame electronic shutter and DVS circuits within the pixels (Brandli *et al.*, 2014). These abrupt changes in firing frequency propagate throughout the network. The variability of DVS event rates differs strongly from the rather uniform spike distribution observed when using Poisson or analog input. In the previous subsubsection, we argued that temporal spike patterns in the test phase can have a detrimental effect because of the asymmetric spike generation mechanism: neuron activity due to a burst of excitatory input cannot be reversed by a later burst of inhibitory input. We can not expect the network to be able to cope with temporal structure in the input which it has never experienced during training.

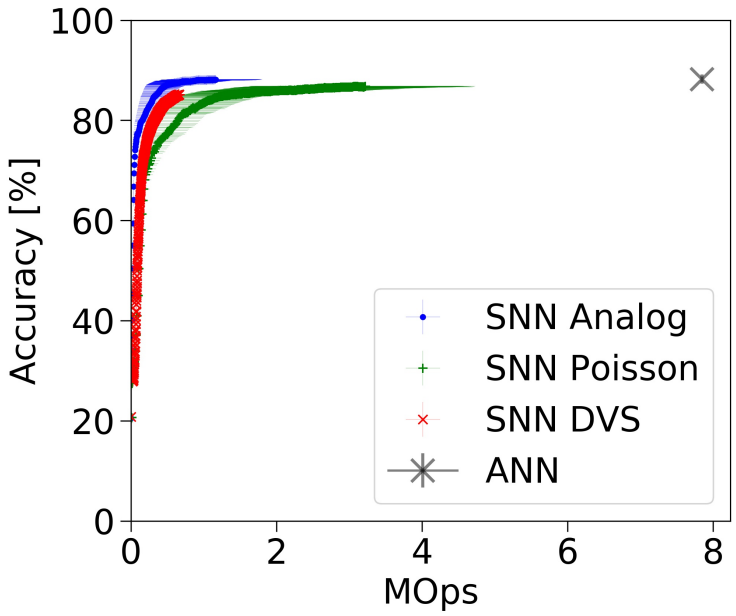


**Figure 4.7:** Average classification test accuracy of ANN (single cross) and SNNs (curves) for DVS, analog and Poisson input, plotted against the number of operations.

This intuitive explanation of the remaining accuracy loss can be validated by using Poisson or analog input, which features constant firing rates as during training. With 88.12% accuracy, analog input nearly closes the accuracy gap. With 86.77% accuracy, Poisson input falls between analog and DVS input, which is reasonable given the variance inherent in a Poisson process.

Figure 4.6 compares the spike trains generated by the different input types. **Analog input** (first row) results in the most regular firing dynamics and the highest support for the correct class label (second neuron in output layer). Surprisingly, the slight variations induced by **Poisson** variability (second row) reduce the network's confidence in the correct class label significantly. Asynchronous **DVS input** (third row) exhibits temporal structure that is not present in analog or Poisson input. The spike rates are generally lower, which accounts for the reduced operation cost, but contributes also to the increased classification error. DVS input with sum-subsampling (bottom row, cf. Sec. 4.2.3.2) contains spike bursts that are fed into the network in close succession, causing the spike train to spread out over a longer simulation time. These temporal patterns – unseen during training – cause the network to confuse the correct output label (right column).

#### 4.2.3.3 *Operation Cost*

Besides classification accuracy, a second important metric for SNN performance is its operation cost. An "operation" for the SNN is defined as a synaptic update, i.e., the update of a neuron's state due to a spike in the preceding layer. This operation corresponds to a simple "addition", in contrast to more costly multiply-accumulate operations needed in conventional ANNs. We compute this quantity

from the network architecture and the number of spikes that each neuron fires during the simulation (Rueckauer *et al.*, 2017).

Figure 4.7 compares the accuracies and operation costs of the ANN trained with L2-regularized biases on max-subsampled data, and of the converted SNN tested on the original DVS events, synthesized analog frames, and Poisson spike trains. The operation cost for the ANN is a single value, because inference consists of a single forward pass. In the SNN, a continuum of classification accuracies is obtained as simulation progresses and more operations are invested. Table 4.2 lists the final accuracy and operation cost at the end of each of the SNN curves.

Summarizing the results, the SNN with analog input provides the highest accuracy while reducing the number of operations by 7x compared to the original ANN. The SNN with DVS input suffers from an accuracy loss of 3%, but compensates for it by a 12x reduction in computational cost.

### 4.2.4 Conclusion

In this work, we explored SNNs as efficient replacements of conventional frame-based ANN on the task of a robot pursuing a moving target. The underlying data set stems from a DVS, which provides a continuous stream of asynchronous events. These event streams are seldom used directly as input to deep neural networks; instead, the events are commonly binned into frames, on which the network is trained and tested. While this frame-based approach grants easy access to a wealth of powerful deep learning frameworks, one sacrifices the advantage of very low latency and potentially sparse computation inherent in asynchronous event streams from a DVS. Converting a pre-trained frame-based ANN into an event-driven SNN aims to combine the best of both worlds: Frame-based training provides us with a high-accuracy model, while inference is done on sparse asynchronous events.

This study confirms earlier findings (Rueckauer *et al.*, 2017; Zambrano *et al.*, 2016) showing that the converted SNN achieves equivalent classification accuracy as the original ANN when using static frames or Poisson spike trains as input. However, we take this analysis a step further by applying the original DVS events as input to the SNN. Initial classification results were close to chance level, indicating significant distortions when transitioning from a synchronously trained model to an asynchronously tested model. As causes for this accuracy loss we identify (1) the way that the training frames are generated from DVS data, (2) extreme weights or biases, and (3) temporal structure present in the asynchronous test data but not in the training frames.

To solve these issues, we propose (1) training frame generation steps that are applicable to the DVS test data as well, thereby minimizing the discrepancy between training and test set, and (2) L2-regularization during training to effectively prevent dominating model parameters. The resulting SNN achieves classification accuracy close to the original ANN. The third issue, temporal structure in the DVS event stream, can only partly be removed, and likely accounts for the remaining 3% accuracy gap between synchronous ANN and event-driven SNN.

By evaluating the computational cost of the SNN when run on DVS events, we confirm the expected improvement in terms of low latency and sparse, change-

**Table 4.2:** Classification accuracy of the original ANN and the converted SNN.

| Model (input type) | Accuracy [%] | Operations [MOperation (Op)] |
|---|---|---|
| ANN (analog) | 88.25 | 7.85 |
| SNN (analog) | 88.12 | 1.15 |
| SNN (Poisson) | 86.77 | 3.06 |
| SNN (DVS) | 85.19 | 0.66 |

driven operation. Specifically, inference in the SNN can be done using 12x less computations on this data set. Further, the computations consist of simple additions, which are energetically cheaper than the multiply-accumulate operations used in conventional ANNs. Future work concerns the measurement of the energy consumption (including the cost of memory transfer due to keeping neuron states).

To close the remaining accuracy gap, extensions of this work might consider training on the DVS events directly. This would create a purely asynchronous setting and potentially enable the model to accurately process streams with highly variable event rates. Regardless of the training method, detrimental inhomogeneities in the DVS input could potentially be removed by low-pass filtering or other preprocessing with smoothing effect.

The present work is a step towards efficient inference in mobile and embedded systems requiring low latency and computation cost, which will in particular profit from development of asynchronous event-based computing hardware.

### 4.2.5 Appendix

The study above applied DVS input to the predator-prey task. After concluding the corresponding paper, we extended the experiments to another event-based vision task, Roshambo (Lungu *et al.*, 2017). In this section we describe the dataset and training procedure, and report the accuracy-operations tradeoff for the converted SNN. We demonstrate the computational advantage of directly using DVS events rather than constant input currents, and compare to a dedicated CNN-accelerator called 'NullHop' (Aimar *et al.*, 2019). We find that the SNN reaches the same level of accuracy with a quarter of the operation count needed by the original model, and half the number of operations needed by the CNN accelerator.

#### 4.2.5.1 *Dataset*

This application consists of a hand symbol recognition robot that plays the game of rock-paper-scissors. A network is trained to detect four classes (rock, paper, scissors, and background). With less than 10 ms latency and 99.9% accuracy, the DVS+CNN creates the illusion that the machine always outguesses the human.

The dataset was collected using the DAVIS event-based camera (Lichtsteiner *et al.*, 2008) and jAER, a software package developed to process DAVIS data. Continuous, labeled recordings were produced for each of the three symbols used in the game

of Rock-paper-scissors, as well as for a fourth input category consisting of various non-hand images, such as other body parts and furniture. The resulting recordings were converted into AVI movies by accumulating asynchronous DVS address-events into $64 \times 64$ pixel 2D histograms of a constant total number of events. The movies were then cut into individual frames and compiled in an LMDB database along with their labels (rock, paper, scissors or background). 15 participants were each recorded for (3 symbols) $\times$ (2 hands) $\times$ (2 minutes), resulting in around 1.3 million images for each of the four classification categories. Frames were generated following the reasoning outlined in Sec. 4.2.2.2: All DVS events were rectified to be positive irrespective of the sign of the luminance change and each resulting image pixel was limited to a 200-event maximum grayscale bin value. Moreover, the resulting pixel values were mapped to a 0-1 range by performing a $3\sigma$ normalization.

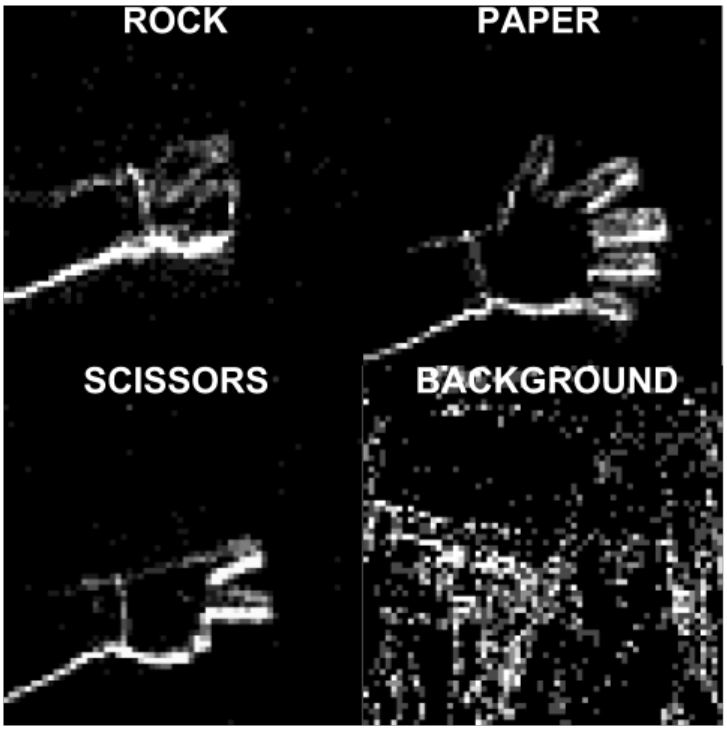


**Figure 4.8:** Examples of the four classes of the rock-paper-scissors game, using constant count DVS frames.

#### 4.2.5.2 *ANN Training*

A 5-layer rate-based convolutional neural network was trained to perform gesture recognition on the collected data (Lungu *et al.*, 2017). The CNN architecture consists of 5 convolutional layers, each using the ReLU activation function and followed by 2x2 max pooling. Kernels are square with dimension 5, 3, 3, 3, 1 respectively. The network has 114k parameters, needing 18 MOp to classify one image. 40 training epochs were performed in Caffe, requiring about 5h on an Nvidia GTX980 GPU. Two data augmentation methods were used while training: random image mirroring

and variable event numbers used to create the histograms. These events ranged from 500 to 4000 for one frame.

#### 4.2.5.3 *Results*

For conversion we follow the same procedure as described in Sec. 4.2.2.4. The SNN is then evaluated on the testset (10% of all available frames), using various input modalities. As reference, we take the original ANN, which achieves an accuracy of 99.85% at 18 MOps.

Figure 4.9 shows the performance of the SNN using the DVS events (labeled 'DVS' in the Figure). The blue dashed-dotted line ('Analog') takes as input the image-frames that are used for testing the ANN, and feeds the analog value at each pixel to the first layer neurons in the form of a constant input current. Equivalently, one can use these analog pixel values to elicit Poisson spike trains (light green triangles, 'Poisson'). If the Poisson rates are high enough, they faithfully approximate the original analog values. However, to allow a fair comparison to the DVS scenario, we can restrict the number of Poisson spikes to the amount used by the DVS input, namely 2000 events per sample. The result is labeled 'Poisson 2000' (dark green triangles). As seen in the Figure, the reduced number of input events plays in favor of the operation count without impeding classification accuracy. A similar picture develops when using the DVS events in the manner described above, though the convergence to the optimal error rate appears to be slightly slower than for the Poisson input. The reason is that the DVS event sequences contain temporal structure that was averaged out when constructing the binned frames on which the ANN was trained, as shown in Sec. 4.2.3.2. Poisson input is able to better reproduce this mean firing-rate representation of the ANN. However, using DVS events directly does not cause as big a drop in accuracy as for the predator-prey task studied above.

For comparison, we also show the original ANN model ('ANN', brown cross), which achieves 99.85% accuracy with 18 Million operations. The low-precision zero-skipping model 'NHP' (yellow cross) achieves the same accuracy with only 9 Million operations because redundant operations are skipped by the CNN accelerator hardware. The SNN (converted from 'ANN') reaches the same accuracy at about the same number of operations as 'NHP', when using the dense analog frames as input, or Poisson spike-trains with high rates. In contrast, we can further reduce the number of operation by a factor 2 when using the original DVS events or an equivalent amount of Poisson spikes as input. Both 'NHP' and SNN exploit spatial sparsity by skipping zeros in the feature maps, but the SNN is additionally able to exploit temporal sparsity and obtain accurate classification results from a small number of input events.

#### 4.2.5.4 *Conclusion*

This experiment highlights the benefit of using data-driven updates from event-sensors such as the DVS, which enables a continuous improvement of the classification output as more events flow in. Potential use cases are computationally efficient

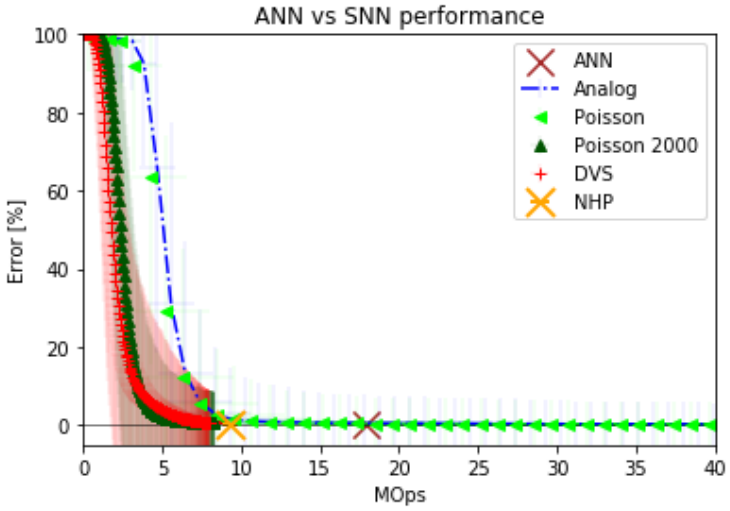


**Figure 4.9:** The two crosses mark two ANN architectures: The original model 'ANN', and the low-precision zero-skipping architecture 'NHP' on the NullHop CNN accelerator. The four curves represent the SNN (converted from the original 'ANN'), with four different input types. See text for details. While the NullHop version halves the number of operations needed to achieve the best classification accuracy, the SNN with Poisson input gets to within 1% of the ANN accuracy using only a quarter of the original operations.

SNN models with low latency that can run on dedicated neuromorphic hardware platforms, e. g. for the face detection, pedestrian detection, or keyword spotting.

## 4.3 RETINA STIMULATION

In the previous sections we demonstrated the usefulness of event-based sensors like the DVS in image processing with SNNs, compression of video streams, and computation of optical flow. In this section we turn to a real biological system, the mouse retina, and study the potential use of a sensor like the DVS in retinal prostheses.

### 4.3.1 Introduction

Retinal prosthetics require low-power, low-latency components. The DVS features microsecond temporal resolution, high dynamic range, and sub-millisecond latency at low energy consumption. In a collaborative study between Sensors Group in Zurich and the Vision Science Lab at Chungbuk National University in South Korea, we investigated the potential of applying the DVS to aid patients with retinal degeneration. To evaluate whether the high temporal resolution of the DVS can be leveraged with multi-electrode stimulation, we injected pairs of electric pulses of varying delay into wildtype and rd1 mouse retinal ganglion cells in-vitro, using a 60 electrode MEA. We then determined the stimulus protocol that optimizes temporal resolution and cell response. Finally, we established an interface between DVS and MEA, and demonstrated for a single pixel that cells can be driven by DVS events with millisecond precision.

### 4.3.2 Experiment 1: Time Resolution of MEA Stimulation

The DVS signals log-intensity changes with microsecond precision, which would allow for unprecedented temporal resolution in a retinal prosthesis. However, precise electric stimulation using a MEA is challenging because of the spatio-temporal overlap of stimulation pulses at different electrodes or points in time. In the setup used here, electrodes are spaced with 200 $\mu$m, and a typical stimulus of 1 ms duration and 10 $\mu$A amplitude applied in one corner of the MEA may elicit a weak spike response even in cells at the opposite corner of the quadratic array.

To determine the temporal resolution of RGC activity in response to MEA stimulation, we devised an experiment that for simplicity does not yet involve the DVS. The MEA used here was manufactured by Multichannel Systems GmbH Reutlingen, Germany, and consists of an $8 \times 8$ array of electrodes spaced with 200 $\mu$m. The four corner slots are not used; electrode 15 constitutes the reference node. Out of the 59 remaining channels, we used number 57 and 71 for stimulation, the others for recording the cell responses (c. f. Fig. 4.10 for a layout of the MEA.)[3] Both stimulation electrodes were programmed to send a bipolar cathodic-first current pulse of 1 ms duration and 10 $\mu$A peak-to-peak amplitude repeated with a frequency of 1 Hz. After every 40 repetitions, the pulse at ch57 is delayed relative to ch71 by an additional 5 ms, for a total of 50 ms at the end of the experiment. Figures 4.10

3 The criteria for selecting channels 57 and 71 were that they should be well attached to the tissue in order to elicit a strong response in neighboring cells, but also located far apart to minimize spacial interference.

illustrates the stimulation protocol and shows an example of a recorded voltage trace.

The raw voltage trace recorded at each electrode typically combines spikes and local field potentials from several cells simultaneously. A standard procedure in electrophysiology, *spike sorting*, takes care of separating these individual cell activities, using a pipeline of spike detection, filtering, feature extraction and clustering. We extended an open-source spike sorting tool called "tridesclous"[4] for our workflow to automate many manual steps involving data loading, conversion, preprocessing and visualization. This modified software is able to process all channels in parallel within few minutes, enabling the experimenter to make adjustments to the setup at the beginning of the in-vitro experiment. Our source code is available online[5].

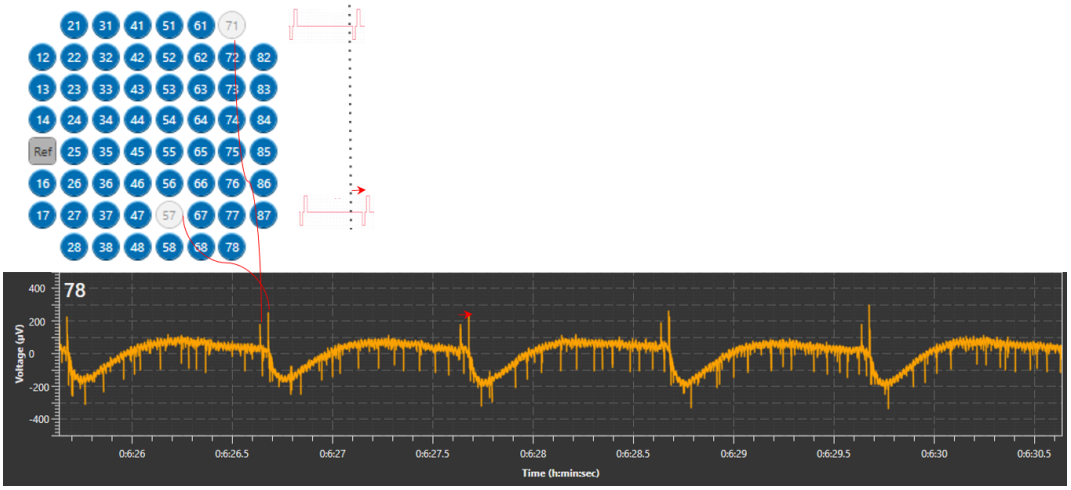


**Figure 4.10:** A) Layout of the MEA used here. The channel numbers indicate the coordinates on the grid. Blue channels are used for recording; white channels (57 and 71) for stimulation. B) Illustration of the stimulation protocol. C) Example of the raw voltage trace recorded at channel 78. The two periodic up-strokes are stimulus artifacts; the first and smaller coming from the more distant stimulation electrode 71, the second and larger from the nearby electrode 57. The delay between the two artifacts corresponds to the delay between applied stimulation (B)). Down-strokes in the voltage trace mark action potentials.

The purpose of this experiment was to determine down to which relative stimulation delay we could clearly distinguish the spike response. Starting from the raw voltage trace recorded at 57 channels, this analysis workflow included the following steps:

1. We first identified the recording channels that contained useful data, i. e. where the retina patch was attatched well and the signal-to-noise ratio was high. This selection was based on inspecting the raw data in Multichannel Analyzer (a tool provided by the MEA manufacturer), as well as the waveforms extracted from a subset of the data using the spike sorter, and the PSTH plots.

4 https://tridesclous.readthedocs.io/, accessed Aug. 2020
5 https://github.com/rbodo/tridesclous

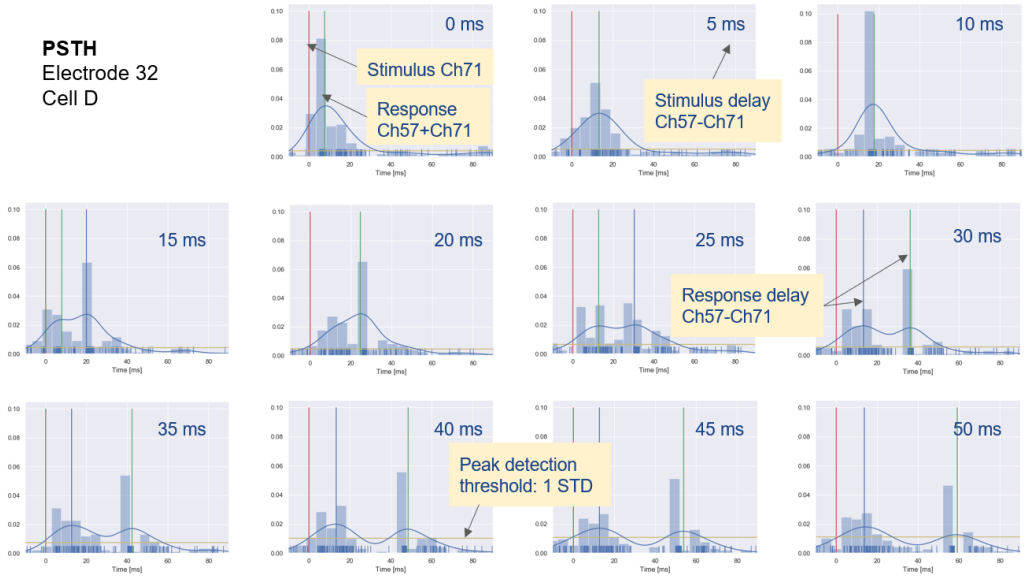


**Figure 4.11:** Peri-Stimulus Time Histogram (PSTH) plots for one of the cells recorded at channel 32. Each panel shows the spike response of the cell accumulated over 40 trials, for one particular delay value (indicated by the label). As the delay between the stimulation pulses at channel 57 and 71 increases, the cell is seen to respond with two distinct peaks.

2. For each of the useful electrodes, we then performed the spike sorting on the full duration of the experiment, while taking special care of removing the stimulus artifact (Automatic filtering of the stimulus artifact is still an open research question). We obtained spikes from 150 cells over the course of two trials (two different pairs of electrodes for stimulation on a patch of retinal tissue).

3. Based on the extracted spike times, we generated the PSTH plots for each cell and for each delay value. We performed a grid-search to determine the optimal PSTH bin size (5 ms) and peak detection threshold ($1 \times \sigma$), using the MSE to the ground truth as metric. Figure 4.11 shows the PSTH plots of one of the cells recorded at electrode 32, for each of the 11 delay values.

4. Using scipy's peak detection algorithm, we measured the peak locations in the PSTH plots, which represent the times of maximum cell response. These peak response times are indicated by vertical lines in Fig. 4.11. One peak is generated in response to stimulation at electrode 71, the other in response to electrode 57. The two peaks overlap when the stimulus delay is small but start to separate at around 15 ms in the example shown here.

5. Given the peak response times, we can plot the response times as a function of the stimulus delay. We expect to see two clusters emerging: One cluster constant over time for the response to ch71, the other cluster linearly increasing with the stimulus delay of ch57. This behavior can indeed be seen in Fig. 4.12. We applied k-means to separate these two clusters and associate them with the two stimulus electrodes. A line is fit to both data groups (dashed lines in

Fig. 4.12). By construction we know that one should have zero slope, the other unit slope. This ground truth is indicated by the solid lines in Fig. 4.12 and matches well the response time fit.

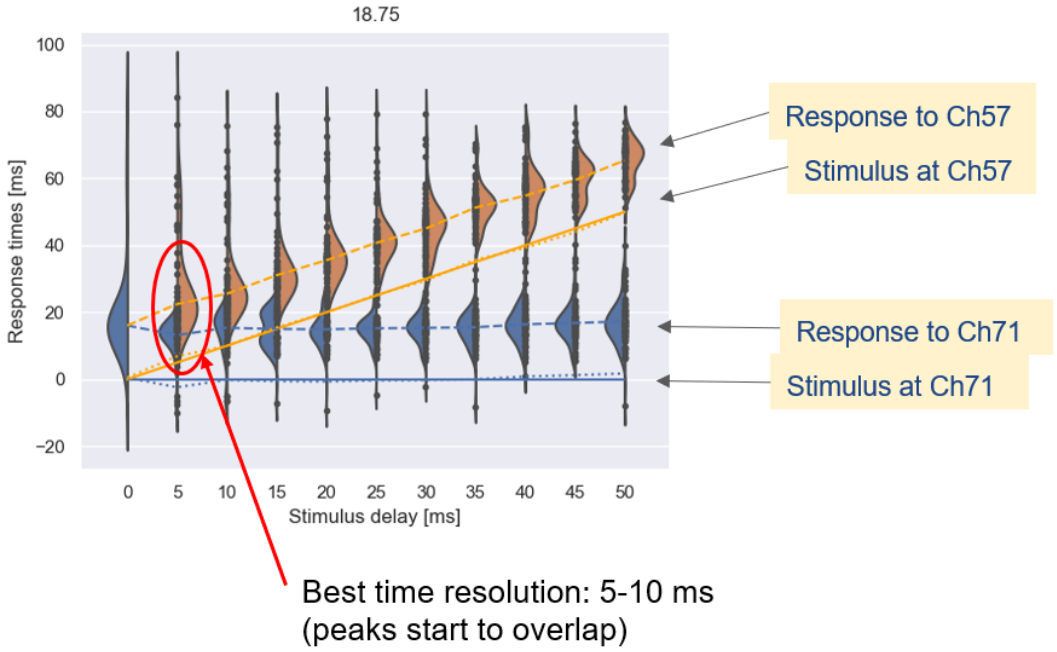


**Figure 4.12:** Response time plotted against stimulus time. As the stimulus delay increases, two clusters start to emerge, one corresponding to the reference channel 71, the other to channel 57.

The first result obtained from this experiment is that the response time measured with the workflow above closely follows the stimulus time as we increase the delay between the two electrodes. This result is little surprising but validates the analysis pipeline. The main finding is how well we can distinguish the two clusters as the delay decreases. In our experiment, such a separation is possible down to the lowest delay tested here, namely 5 ms. A follow-up experiment should devote more attention to even shorter delays. However, the distribution of peak times already shows a significant overlap at this point, which will likely impede a clear distinction at shorter delays. Further, the pulse width of the stimulus itself (1 ms here) sets a limit on the temporal resolution that can be measured with the MEA as the cell response is hidden by the stimulus artifact during the pulse itself (Ahn *et al.*, 2017).

### 4.3.3 Experiment 2: Optimizing the Stimulus Protocol for Temporal Resolution

The previous experiment was performed with a fix stimulus of 10 $\mu$A amplitude and 1 ms width. This pulse shape was optimized to elicit a strong overall spike response (Ahn *et al.*, 2015; Goo *et al.*, 2011; Ryu *et al.*, 2017). However, a large amplitude increases potential spatial interference with other stimulus electrodes, whereas a long pulse width increases temporal interference between simulus artifact and measured cell responses. Intuitively, a stimulation protocol which is sensitive to

spatio-temporal resolution like the one in Sec. 4.3.2 would benefit from shorter and smaller pulses. As a step towards quantifying this intuition, we investigated whether we could optimize the stimulus parameters to minimize the time between a pulse and the cell response that can be measured with the MEA.

In a meta-study based on experimental data from the past 10 years recorded by the Vision Science Lab of Prof. Yong Sook Goo, we analyzed the spike response time of retinal ganglion cells under various stimulus conditions, which span a three-dimensional parameter space defined by triplets [pulse height, pulse width, polarity]. The workflow follows the steps described in Sec. 4.3.2: After identifying usable channels, we applied the spike sorting pipeline to extract individual cell responses for a given simulus shape. These responses were aggregated into PSTH plots, from which we measured the peak response time. These times define hyperplanes over the three dimensional parameter space. For better visualization, we collapse the pulse height dimension as well as the polarity parameter (which is binary: cathodic- vs anodic-first). The result is shown in Fig. 4.13 for wildtype and Fig. 4.14 for rd1 mice.

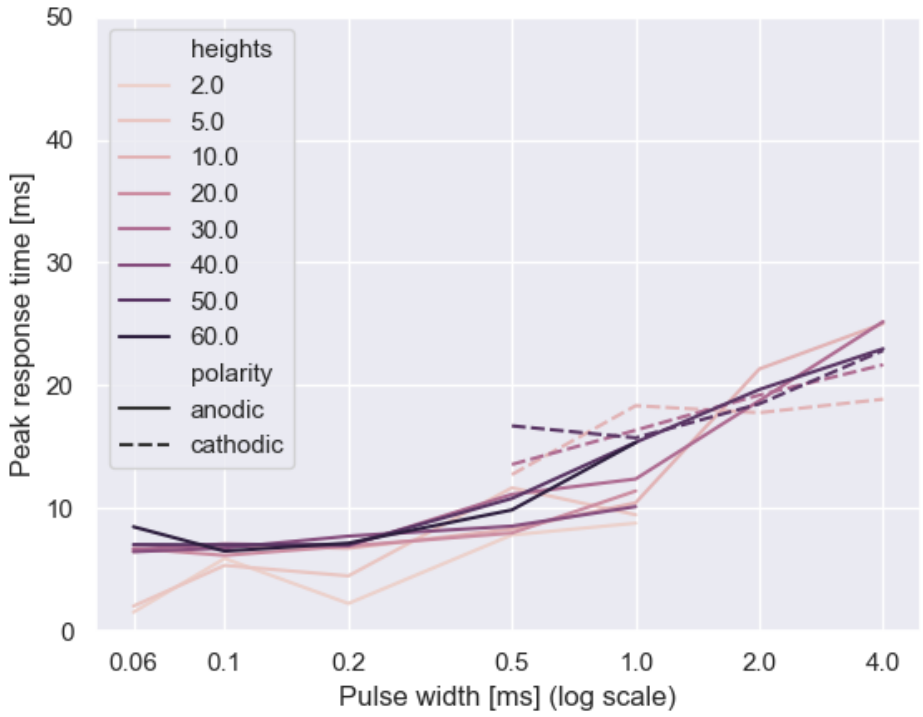


**Figure 4.13:** Spike response time as a function of the stimulus parameters pulse width, height, and polarity. Action potentials were recorded from retinal tissue of wildtype mice.

For RGCs of wildtype mice, we observe a clear increase of the response delay with pulse width. On the other hand, the spike response appears to depend at best weakly on pulse height and polarity. In the critical region of short pulses ($< 0.1$ ms), pulses with low amplitude seem to respond with only millisecond latency.

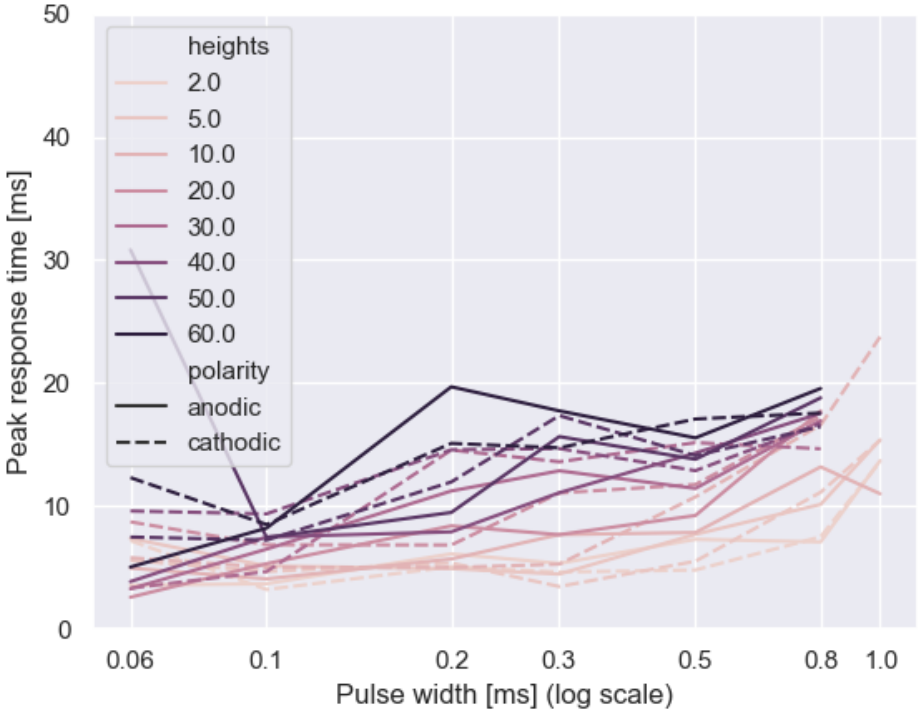


**Figure 4.14:** Spike response time as a function of the stimulus parameters pulse width, height, and polarity. Action potentials were recorded from retinal tissue of rd1 mice.

Further investigation is warranted to rule out attribution of this measurement to the stimulus artifact.

In mice with degenerated retinal tissue, the response delay likewise increases with pulse width. However, we now also observe an increase in response time proportional to the pulse height. Polarity still shows no notable effect. Another difference to wildtype mice can be seen in the regime of short pulses: Here the response time is less well defined, which can likely be attributed to the decreased ability of rd1-cells to respond to stimuli of such short duration.

These results suggest that the time resolution of an MEA stimulation as in Sec. 4.3.2 could be reduced by at least a factor 2 by shortening the stimulus pulse width, possibly further by also decreasing the amplitude, both for wildtype and rd1 mice.

### 4.3.4 Experiment 3: MEA Stimulation from DVS Events

In our final experiment, we implemented two approaches to use DVS events for stimulation of mouse RGCs in vitro, one based on software tools, the other on hardware. The software approach, though considerably more flexible, turned out to be infeasible with the tool versions available at the lab. The hardware pipeline was validated in a simple proof-of-concept application.

#### 4.3.4.1 *Software-Based Pipeline*

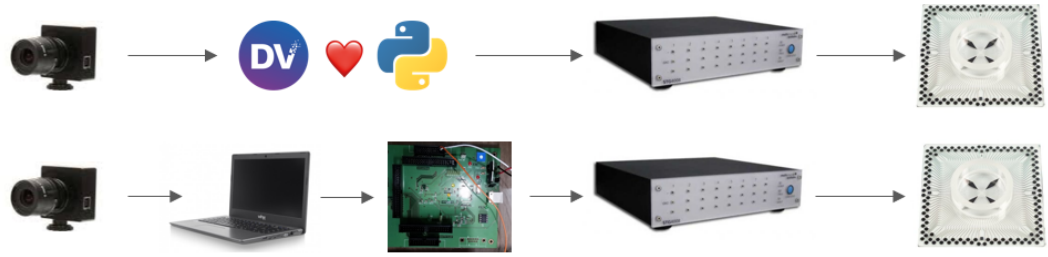


**Figure 4.15:** Pipeline for MEA stimulation from DVS events based on software tools (top) and hardware tools (bottom).

In the software approach, a DVS sensor is connected via Universal Serial Bus (USB) to a host computer, such as the lab PC controlling the MEA. The host PC interfaces to the STG device supplied by Multichannel Systems GmbH, which in turn connects to the MEA itself. Thus, the physical setup only requires plugging in the DVS in the familiar way.

In terms of software infrastructure, the DVS stream is processed by one of the existing address-event frameworks such as jAER, pyAER or the most recent Dynamic Vision (DV) library. We chose DV in part because of its powerful but easy-to-use python binding "DV-Python", and the support and active development by iniVation. After spatio-temporal filtering for noise and resolution reduction, the events are streamed from the host PC to the STG using a python wrapper[6] around the STG driver[7]. At the STG, each event then triggers a predefined electrical stimulus in the MEA. Figure 4.15 illustrates the pipeline.

The advantages of this software-based approach are that it provides an end-to-end pipeline for real-time operation with no overhead for the physical setup and little custom code. In addition, one has full flexibility of processing events and programming the STG, either in python or using the DV and STG Graphical User Interfaces (GUIs). Unfortunately, the STG driver is only available for hardware version 4000 and newer, which made it incompatible with the one present at the lab.

#### 4.3.4.2 *Hardware-Based Pipeline*

Without the ability to program and trigger the STG via software, we developed an alternative that makes use of the digital input ports on the STG to trigger predefined stimuli. As this approach does not support live-streaming the events, a DVS signal is pre-recorded and passed through a spatio-temporal filtering stage in jAER. Then a USBAERmini2 monitor-sequencer board (Berner *et al.*, 2007) is employed to send Transistor-Transistor Logic (TTL) pulses to the STG. Each pulse triggers a predefined electrical stimulus in the MEA, same as in Sec. 4.3.4.1.

One of the functions of the USBAERmini2 board is to sequence previously recorded events from jAER, which could in principle be used to trigger stimuli

6 https://github.com/pyreiz/ctrl-stg4000

7 `McsUsbNet.dll` provided by MEA manufacturer (MCS GmbH)

at the corresponding MEA electrodes. However, the output pins on the board are incompatible with the four available digital input ports on the STG. As a first simple implementation we shortened the request and acknowledge pins of the sequencer, which results in a brief pulse of a few nanoseconds duration whenever an event is sequenced. One clear limitation of this approach is that the address information is lost, only the event timing is preserved, which is however sufficient for our proof-of-concept.

Because the STG requires TTL pulses of at least 20 $\mu$s width, we built a pulse-extender circuit to increase the pulse-width of events sent from the USB monitor-sequencer board to the STG. The advantage of the hardware-based approach is that with TTL pulses, the latency between receiving the trigger and starting the stimulus is less than 0.1 ms, whereas the software-based approach (using USB connection) has a latency of about 1-2 ms.

The downsides of this approach include the substantial overhead to set up the software and hardware infrastructure (including two custom boards). Further, the experimenter enjoys less flexibility in processing events and programming the STG. For instance, only four digital input ports are available to trigger stimuli at the STG, which limits the number of electrodes that can be controlled independently. (One could set up a binary code to address up to $2^4$ out of the 59 available channels). The limitation on the end of the USB board is even more severe in our current approach because we can only transmit timestamp information using the req/ack pin on the USBAERmini2. A possible extension could instead access individual address pins from the sequencer, at the cost of additional logic. Finally, unlike in the software-based approach, jAER currently does not support online streaming from DVS to STG via USBAERmini2. To achieve this, one would have to start by extending the `AEViewer.sequenceFile` class in jAER to retrieve event packages from the filter used for pre-processing. These improvements were beyond the scope of the 3-month exchange program, considering also that the software approach already provides an elegant solution.

#### 4.3.4.3 *Proof-of-Concept Application*

To validate our interface between DVS and MEA, we recorded a visual stimulus of two minutes consisting of full-field flashes at 1 Hz from a computer monitor with the DVS. The DVS events were filtered for noise and spatiotemporally subsampled in jAER. The resulting events were sequenced by the USBAERmini2 board, which generates a TTL pulse for each event. The pulses are passed via the pulse extender to the STG, where they trigger bipolar current pulses at channel 57 in the MEA. A picture of the experimental setup can be seen in Fig. 4.16.

We measure the elicited spikes of the RGCs in response to the DVS stimulation at 1 Hz. The subsequent analysis follows the first steps listed in Sec. 4.3.2. We obtain spikes from about 30 cells over the course of three trials (three different stimulus electrodes on one patch of retinal tissue). From the PSTH plots we determine the response time of cells after a stimulus pulse. We then plot the correlation between trigger times and cell response times (Fig. 4.17 left) and find perfect correlation (Pearson coefficient of 1) - the cells respond correctly to DVS stimulation.

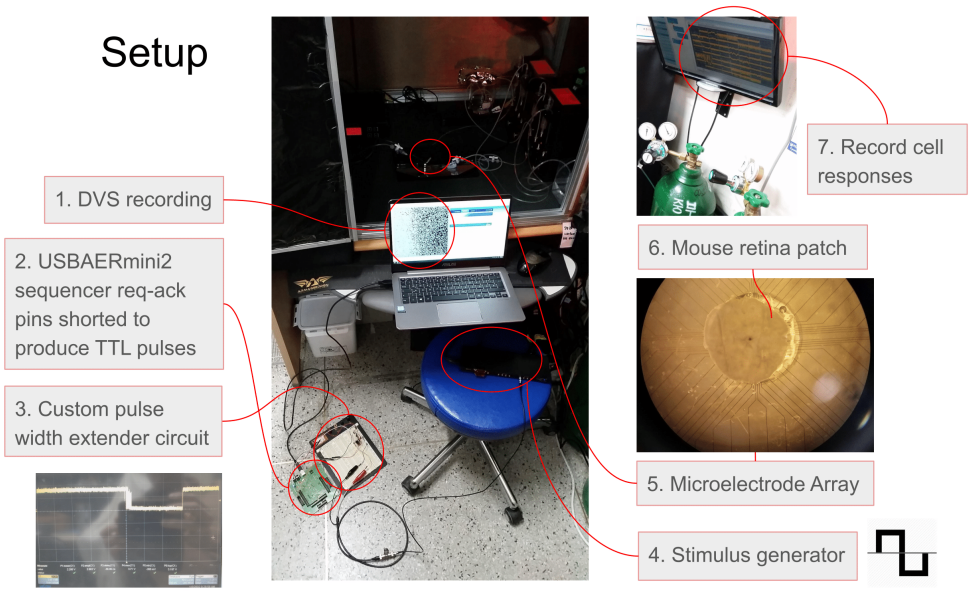


**Figure 4.16:** Experimental setup for MEA stimulation from DVS events.

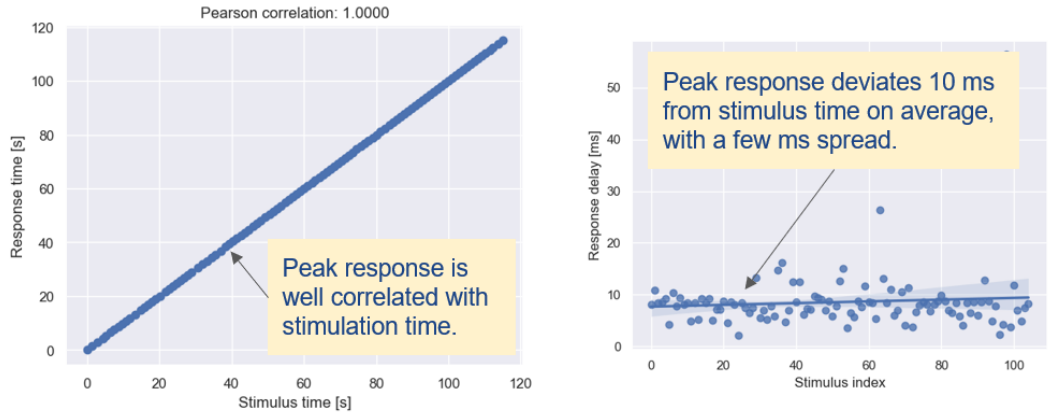


**Figure 4.17:** Response correlation and timing jitter.

As a last step, we plot the jitter of response times around the ground truth (the diagonal in Fig. 4.17), to determine the temporal precision with which the cells respond to DVS stimulation. From Fig. 4.17 B, we see that the response time follows the stimulus time by about 10 ms on average, with few milliseconds spread.

### 4.3.5 Discussion

To gauge to potential of using the DVS in the context of retinal prosthetics, we investigated the temporal resolution of cell responses that can be achieved with MEA stimulation. We showed the minimum time resolution to be lower than 5 ms. In a follow-up study, we found that the response latency could be further decreased by reducing the stimulus pulse width (both in wildtype and rd1 mice) and amplitude (in rd1). These conclusions are limited by relying on PSTH peak analysis, which is

susceptible to interference with the stimulus artifact. Patch-clamp measurements would be more precise. However, the purpose of the present study was not to show how fast cells respond in principle (which has been done before (Hadjinicolaou *et al.*, 2015; Im *et al.*, 2016; Jalligampala *et al.*, 2017; Jepson *et al.*, 2014; Lee *et al.*, 2013)), but to determine the response precision (latency and jitter) in case of micro-electrode stimulation.

Finally, we showed a proof-of-concept of using the DVS to drive RGCs. The hardware-based approach is currently very limited; a flexible software approach can be used if a more recent MEA is available.

### 4.3.6 Conclusion

This 3-month research project in collaboration with Chungbuk National University developed the tools to stimulate retinal ganglion cells using a DVS camera, and analyze the cell response. We obtained first experimental results that demonstrate millisecond precision of the elicited spike response, which is critical for better subjectively perceived visual recognition (Greene 2007). The software is made freely available to perpetuate the impact of this short project and to benefit future work in this promising research direction.

# 5 DYNAMIC FEATURE MASKING

Sequential data such as video are characterized by spatio-temporal redundancies. As of yet, few deep learning algorithms exploit them to decrease the often massive cost during inference. This work leverages correlations in video data to reduce the size and run-time cost of deep neural networks. Drawing upon the simplicity of the typically used ReLU activation function, we replace this function by dynamically updating masks. The resulting network is a simple chain of matrix multiplications and bias additions, which can be contracted into a single weight matrix and bias vector. Inference then reduces to an affine transformation of the input sample with these contracted parameters. We show that the method is akin to approximating the neural network with a first-order Taylor expansion around a dynamically updating reference point. For triggering these updates, one static and three data-driven mechanisms are analyzed. We evaluate the proposed algorithm on a range of tasks, including pose estimation on surveillance data, object detection on Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago (KITTI) driving scenes and ImageNet videos, as well as denoising MNIST digits, and obtain compression rates up to 12×.[1]

## 5.1 INTRODUCTION

Natural video is often characterized by local spatio-temporal redundancies between frames, e. g. in surveillance or freeway driving recordings. DNNs are powerful tools to process image data, but can be prohibitively expensive for use in mobile devices or always-on scenarios. The high degree of correlation between consecutive samples in sequential data calls for exploitation in DNNs.

To capitalize on this correlation for a more efficient solution to video processing, we previously presented a method to linearize part of a DNN or an entire DNN around a dynamically updating reference point. This dynamic updating is similar to the evaluation of a first-order Taylor expansion of a network function around an operating point (Rueckauer *et al.*, 2019b). By combining the linearized parts of the DNN, we arrive at a compressed DNN which has fewer neurons and parameters, and therefore also reduced operations during inference.

This network approximation method draws upon two common characteristics of DNNs: 1) The composite structure of neural networks where the mathematical function of a set of layers consists of a sequence of matrix multiplications. 2) The piecewise linear nature of the frequently used ReLU activation function.

By replacing the element-wise activation function with a mask vector, a stack of layers can be contracted into a single layer. This layer, represented by a single weight

1 This chapter is based on Rueckauer *et al.*, 2019b and a journal submission currently under review.

matrix, approximates the function of the original layer stack around a reference point. The reference point, along with the contracted weight matrix, can be updated when temporal changes in the input sequence become large. These updates are expected to be sparse in slowly varying scenes. In between updates, inference is done cheaply using the contracted weight matrix.

In Rueckauer *et al.*, 2019b, we apply this compression method to a denoising autoencoder and demonstrate little loss in accuracy but savings in computation. This paper extends the previous work in three ways. First, we evaluate an improved criterion for dynamically updating the activation masks. Second, we discuss in detail the algorithm implementation and the computational cost. Finally, we validate the findings in Rueckauer *et al.*, 2019b by applying this method to a diverse set of tasks, namely, a deep CNN for car tracking, pose estimation on surveillance data, and the YOLO architecture (Redmon *et al.*, 2016) for object detection in two datasets: the KITTI driving benchmark (Geiger *et al.*, 2012) and a subset of the ImageNet object-detection-from-video challenge (Russakovsky *et al.*, 2015).

Even though we demonstrate the applicability of the proposed method on a wide range of tasks and architectures, there are limitations in using the network contraction. For some networks, only a part of the model can be contracted, for instance in the presence of skip-connections. The supported network features are listed in Sec. 5.2.3. Also, when a stack of convolution layers are contracted, they are unrolled (giving up weight sharing) and afterwards replaced by a combined fully-connected layer. As a consequence, when the size of both the input and output of the contracted network section is large, the size of the contracted weight matrix may also be large. Sec. 5.2.4 quantifies this cost, and Sec. 5.3 discusses examples of architectures that are either well suited or less suited for contraction.

### 5.1.1 Related Work

#### 5.1.1.1 *Network Compression*

Various methods have been proposed for making DNNs more efficient in processing sequential data by drawing upon redundancies in the input stream. For example, the authors of the Skip RNN (Campos *et al.*, 2018) employ a control variable for each neuron that is used during training. This variable determines whether the state of the neuron is updated or copied over from the previous time step. A regularization term encourages the model to use a reduced number of state updates during training. This update skipping can be seen as a zero-order approximation: The state is kept constant (copied over). In contrast, our method replaces nonlinearities with masks and because changes can still be propagated through these masks, our method gives a first-order approximation.

Delta RNNs (Neil *et al.*, 2017) reduce the computation in the network by capitalizing on the stability of RNN activation patterns and transmitting neuron activations only when the change in activation across two sequential timesteps exceeds a defined threshold. Change-based CNNs (Cavigelli *et al.*, 2020) are based on a similar principle by exploiting the spatio-temporal sparsity of pixel changes in video data. The resulting accuracy drop from skipping computations is proportional to the

targeted efficiency gain and the difficulty of the dataset. Unlike Cavigelli *et al.*, 2020, our method does not discard small activation changes via a local threshold, instead all changes are propagated through the mask that replaces the nonlinearity.

Other compression methods include the use of low-precision models (Hubara *et al.*, 2016; Mishra *et al.*, 2018; Miyashita *et al.*, 2016), pruning of redundant weights, feature maps or entire layers (Bengio *et al.*, 2016; Han *et al.*, 2015; Lin *et al.*, 2017), and training convolutional architectures with fewer connections (Howard *et al.*, 2017; Iandola *et al.*, 2016). Others train networks to develop input-dependent paths within the DNN (McGill *et al.*, 2017). In our case, network parts are not skipped or removed, but layer stacks are contracted to a single layer by replacing the ReLU nonlinearities with a dynamic mask.

#### 5.1.1.2 *Interpolation Between Key Frames*

Zhu *et al.*, 2017 presented a video recognition framework that runs a DNN on certain key frames and propagates the feature maps corresponding to a key frame to subsequent frames via a flow field. The optical flow is computed with a CNN that is trained end-to-end with the video recognition model. The method is similar to ours in the use of key frames, but differs in how inference is done between key frames (propagating hidden layer states along flow vectors versus linearizing the network function and contracting a stack of layers). The key frames in Zhu *et al.*, 2017 are updated at regular intervals, while we propose various input-driven update predictors. Lastly, their method makes explicit use of the temporal correlation in a video via optical flow whereas we implicitly draw on this assumption to motivate a linear approximation.

#### 5.1.1.3 *Taylor Expansion in DNNs*

We show in Sec. 5.2 that our network contraction method can be seen as a first order Taylor expansion, where the derivatives are taken with respect to the network input. Taylor expansions are also applied to neural networks in other contexts, e. g. to explain nonlinear classification decisions (Montavon *et al.*, 2017), to generate a class saliency map of a specific input (Simonyan *et al.*, 2014a), and to analyze the learning convergence under various optimizers (Balduzzi *et al.*, 2016).

#### 5.1.1.4 *Activation Patterns and Linear Regions in DNNs*

The piecewise linear nature of ReLU-DNNs gives rise to linear regions in the input space, where the gradient of the loss with respect to the input can be shown to be stable, leading to robustness against adversarial attacks (Lee *et al.*, 2019). These linear regions are defined by *activation patterns* (Raghu *et al.*, 2017), i. e. binary masks that encode the on/off state of the ReLU for each neuron in the network. These linear activation patterns have been studied in the context of neuron visualization (Fischetti *et al.*, 2017), of estimating a network's expressiveness by counting the number of linear regions (Montúfar *et al.*, 2014; Raghu *et al.*, 2017; Serra *et al.*, 2018), and in adversarial attacks (Cheng *et al.*, 2017; Fischetti *et al.*, 2017; Weng *et al.*, 2018). To the best of our knowledge, this is the first work which pursues computational

savings by making use of static activation patterns in linear regions of the input space.

## 5.2 METHODS

In Sec. 5.2.1 we review the masking and contraction method as proposed in previous work (Rueckauer *et al.*, 2019b), and extend it with an improved dynamic indicator for mask updates (Sec. 5.2.2). We describe the algorithm implementation in Sec. 5.2.3 and discuss the reduction in the computational cost in Sec. 5.2.4.

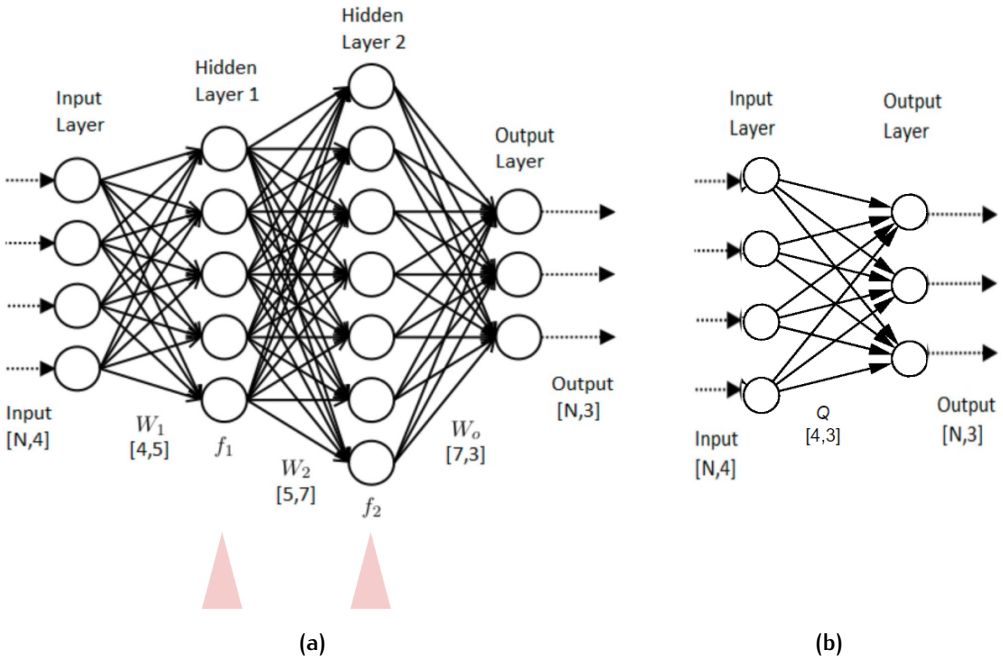


**Figure 5.1:** Illustration of network masking and contraction. (a) Activation functions are first replaced by dynamically updating masks (triangles, cf. Eq. 5.3). (b) Hidden layer weight and mask matrices are then contracted (Eq. 5.5). Adapted from Rueckauer *et al.*, 2019b.

### 5.2.1 Activation Masking and Network Contraction

In this work, we consider feed-forward deep neural networks. Each neuron in a network layer receives as input the linear combination of the outputs of the preceding layer, and applies a nonlinear activation function on this weighted sum of inputs. For instance, an $n$-layer fully-connected ANN has the following computational structure:

$$F(\mathbf{x}) = W^n f\big(W^{n-1} \dots f\left(W^1\mathbf{x} + \mathbf{b}^1\right) + \cdots + \mathbf{b}^{n-1}\big) + \mathbf{b}^n, \quad (5.1)$$

where $F$ is the network output, $\mathbf{x}$ is the network input, pairs $(W^k, \mathbf{b}^k)$ represent the weights and biases of layer $k \in [1, n]$, and $f$ is the nonlinear activation function of a neuron. Typically, the network output is also passed through a final nonlinearity $f_{\text{out}}$, e. g. a softmax in a classification task. To simplify notation in the following equations, we do not explicitly write this output nonlinearity, but imply that it is applied as usual after a forward-pass through the network. A common nonlinearity for hidden layers is the ReLU, which is a special case of the leaky ReLU:

$$f(z_i^k) = \begin{cases} z_i^k & \text{if } z_i^k > 0 \\ \alpha z_i^k & \text{otherwise.} \end{cases} \tag{5.2}$$

The variable $z_i^k$ stands for the pre-activation of neuron $i$ in layer $k$, i. e. the summed output before applying the nonlinearity. The leak parameter, $\alpha$, defines the slope of the left branch and typically has a small value (e. g. $\alpha = 0.1$). In the case of the standard ReLU, $\alpha = 0$.

From (5.2), it is apparent that the element-wise application of the (leaky) ReLU $f$ in (5.1) is equivalent to the element-wise multiplication with a mask $\mathbf{m}^k$:

$$F(\mathbf{x}) = W^n \mathbf{m}^{n-1} \odot \left(W^{n-1} \dots \mathbf{m}^1 \odot \left(W^1 \mathbf{x} + \mathbf{b}^1\right) + \cdots + \mathbf{b}^{n-1}\right) + \mathbf{b}^n \tag{5.3}$$

where $\odot$ denotes the Hadamard product. In this work, the Hadamard product takes precedence over the matrix-multiplications, i. e. the mask $\mathbf{m}^k$ is applied to the pre-activation of layer $k$ before multiplication with the weights $W^{k+1}$ of layer $k+1$. The mask $\mathbf{m}^k$ of layer $k$ is a vector whose length is equal to the number of neurons in layer $k$. Its entries are defined by[2]

$$m_i^k = \begin{cases} 1 & \text{if } z_i^k > 0 \\ \alpha & \text{else.} \end{cases} \tag{5.4}$$

Because matrix multiplications are associative, we can contract any number of matrix-, mask-, and vector-products in (5.3) into a single weight matrix $Q$ and bias vector $\mathbf{q}$ (see also Fig. 5.1a):

$$F(\mathbf{x}) = Q\big|_{\mathbf{x}} \mathbf{x} + \mathbf{q}\big|_{\mathbf{x}}, \text{ where} \tag{5.5}$$

$$Q\big|_{\mathbf{x}} = W^n \text{diag}(\mathbf{m}^{n-1}) \odot W^{n-1} \dots \text{diag}(\mathbf{m}^1) \odot W^1,$$

$$\mathbf{q}\big|_{\mathbf{x}} = W^n \mathbf{m}^{n-1} \odot \left(W^{n-1} \dots \mathbf{m}^1 \odot \mathbf{b}^1 + \cdots + \mathbf{b}^{n-1}\right) + \mathbf{b}^n. \tag{5.6}$$

These expressions are derived by expanding the nested products in (5.3). The mask vector is written in matrix-diagonal form so all multiplicands in $Q$ are matrices.

2 Because the mask depends on the current input sample, it could be written as $\mathbf{m}^k\big|_{\mathbf{x}}$. We omit this subscript here for clarity and introduce it later.

The key idea to reduce the run-time cost of a neural network on sequential data is to use a sample $\mathbf{x}_*$ to compute the $(Q, \mathbf{q})\big|_{\mathbf{x}_*}$ parameters once, and then to perform inference on the next few samples in the sequence by using the much simplified affine transformation (5.5).

This process is related to the linearization of a complicated function (in our case the network $F(\mathbf{x})$) via Taylor expansion around a reference point $\mathbf{x}_*$:

$$T_F(\mathbf{x}) = F(\mathbf{x}_*) + J_F\big|_{\mathbf{x}_*}(\mathbf{x} - \mathbf{x}_*) + \ldots, \tag{5.7}$$

with $J_F\big|_{\mathbf{x}_*} = \frac{\partial F}{\partial \mathbf{x}}\big|_{\mathbf{x}_*}$ the Jacobian of $F$ evaluated at $\mathbf{x}_*$. We determine the conditions under which our contracted network $C_F(\mathrm{x})$, defined by

$$C_F(\mathrm{x}) = Q\big|_{\mathbf{x}_*}\mathbf{x} + \mathbf{q}\big|_{\mathbf{x}_*}, \tag{5.8}$$

resembles a Taylor approximation around the reference point $\mathbf{x}_*$. By construction (c. f. (5.1)), the contracted network exactly equals the original network at the reference point:

$$C_F(\mathrm{x}_*) = Q\big|_{\mathbf{x}_*}\mathbf{x}_* + \mathbf{q}\big|_{\mathbf{x}_*} = F(\mathbf{x}_*). \tag{5.9}$$

By using identity (5.9) to replace the zero-order term in (5.7), and after some rearranging, we obtain:

$$T_F(\mathbf{x}) = \mathbf{q}\big|_{\mathbf{x}_*} + \left(Q\big|_{\mathbf{x}_*} - J_F\big|_{\mathbf{x}_*}\right)\mathbf{x}_* + J_F\big|_{\mathbf{x}_*}\mathbf{x} + \ldots. \tag{5.10}$$

Comparing coefficients of (5.8) and (5.10), we can see that the contracted network is identical to a first-order Taylor expansion if $Q$ is equal to the Jacobian:

$$C_F(\mathbf{x}) = T_F^{(1)}(\mathbf{x}) \quad \text{if} \quad Q\big|_{\mathbf{x}_*} = J_F\big|_{\mathbf{x}_*}. \tag{5.11}$$

In Sec. 5.3, we show that this relation holds through our experiments with a denoising autoencoder.

Having established a relationship between the contracted network and the first-order Taylor approximation of the original network, we now consider when to update the $Q$ matrix at a new reference point $\mathbf{x}_*$. At $\mathbf{x}_*$, the simplified affine transformation (5.5) is exactly equivalent to a standard forward-pass through the original multi-layer network. As the input samples begin to deviate from the reference point, the masks $\mathbf{m}$ used for computing the contracted $(Q, \mathbf{q})\big|_{\mathbf{x}_*}$ parameters remain accurate only under the condition that the neurons do not change the sign of their activity $\mathbf{z}$ (c. f. (5.4)). For instance, if a neuron $i$ in layer $k$ changes from being positively activated at sample $\mathbf{x}_* = \mathbf{x}_t$ to being negatively activated at sample $\mathbf{x}_{t+1}$, then the corresponding entry in the mask vector $m_i^k$ should be $\alpha$ instead of 1, and the $(Q, \mathbf{q})\big|_{\mathbf{x}_t}$ parameters introduce an error. Note that such errors occur exclusively at sign changes: Otherwise, the $(Q, \mathbf{q})\big|_{\mathbf{x}_t}$ parameters accurately encode the network function even under arbitrarily large changes of $z_i^k$.

To keep the network operation accurate even when activation signs are changing over the course of presenting an input sequence, it is necessary to recompute

the $(Q, \mathbf{q})$ parameters at appropriate intervals, using updated binary masks that represent the state of network activations at the new reference point. Thus, we face an accuracy-efficiency trade-off: The less often the update of the $(Q, \mathbf{q})$ parameters, the lower is the run-time cost of inference, but the higher is the risk of inaccurate activation masks. This trade-off is discussed in Section 5.3. We now turn to the question of how mask updates can be predicted.

### 5.2.2 Indicators for Mask Updates

The masks $\mathbf{m}^k$ needed to compute $(Q, \mathbf{q})$ depend on the current input $\mathbf{x}$ and the resulting activity of intermediate layers. We describe in more detail the three update indicators used in previous work (Rueckauer *et al.*, 2019b), and propose a fourth metric which directly measures the number of activation sign changes.

#### 5.2.2.1 *Regular Frame Update*

The most straight-forward baseline criterion for mask updates is the use of a regular update interval. The hyperparameter in this case is to perform the network contraction after every $n$-th frame (Fig. 5.2a). The advantages of this criterion are that it is easy to interpret and relate to expected savings in computations. Further, it offers a safety guarantee by limiting the longest period without updates to $n$ frames. On the other hand, potential savings due to rather static scenes with more than $n$ frames are missed.

#### 5.2.2.2 *Input MSE*

A more flexible criterion that accounts for direct changes in input is by computing the pixel-wise square difference on input images, i. e. , the mean-square error (MSE) between the current frame and the frame at the time of the last update (Fig. 5.2b). If the MSE value surpasses a given threshold, a mask update is triggered. A potential drawback of this method is that the *MSE input* criterion may be set off by global changes in lighting etc., which are not necessarily related to changes in the object of interest.

#### 5.2.2.3 *Output MSE*

Similarly, we can compute the MSE between the network output for the current frame and the output for the frame at the previous update time. If the MSE exceeds a given threshold (which may vary from the *MSE input* threshold), the masks are updated for the next frame (Fig. 5.2c). A drawback is that the *MSE output* criterion triggers updates for the next sample, i. e. with one frame delay.

#### 5.2.2.4 *Number of Activation Sign Changes*

Activation masks can be applied to all the layers in a network as shown in (5.3), or to a subset of layers. For instance, one can choose to mask and contract only the upper half of a network. In that case, we can apply a fourth criterion to predict mask updates. Mask updates become necessary when individual neurons in a layer

change the sign of their activation. If only the layers from layer $j$ upwards are masked, we can count the number of neurons in layer $j$ that changed the sign of their activations since the time of the last mask update. A mask update is triggered when more than a certain fraction of the neurons changed their activation sign (Fig. 5.2d). Among the update criteria discussed so far, this predictor correlates most strongly with the actual validity of a mask for a given input sample.

In Section 5.3 we evaluate how well each of these four update criteria maintains the accuracy of the network while sweeping the update threshold.

### 5.2.3 Implementation Details

A brief description of the workflow for compiling and using the contracted network is presented here. Given a network architecture, the network (or network section) is first tested for suitability of contraction. Each layer to be contracted must be transformable into a combination of matrix multiplication, bias addition and mask operations. In particular:

- Fully-connected layers require no modification because their weight matrix is already of rank 2.
- Dropout layers can be removed after training.
- Batch-normalization layers can be integrated into the preceding convolution layer after training.
- Activation layers (inserted after fully-connected or batch-normalization or convolution layers) must be transformable into a mask. For instance, piecewise linear functions like the leaky ReLU work, but not the exponential linear unit ELU.
- Convolution layers share weights among neurons, so (5.6) cannot be applied directly. However, the convolution operation can be written in terms of a matrix-vector product by unrolling the convolution using a generalized Toeplitz matrix. In fact, several deep learning libraries (including caffe) utilize an image-to-column transformation to enable convolution via optimized Generalized Matrix-Multiplication (GEMM) methods (Chetlur *et al.*, 2014).
- Transposed convolution layers, as used in autoencoders, can be written as convolution layers, where upsampling is achieved by inserting a mesh of zeros in the layer input.
- Average-pooling layers can be written as convolution layers and then unrolled as such.
- Max-pooling layers can likewise be written as convolution layers, but with a mask prepended, which gates the maximum activation of each pooled patch. This mask is then updated in the same way as the activation masks of ReLU layers.

- Networks with parallel branches, like skip-connections in ResNets or inception modules in GoogLeNet, can be contracted by applying each branch separately on a copy of the input to a multi-branch section.

After the initial checks and layer transformations, the network to be contracted now consists of a set of triplets $(\tilde{W}^k, \tilde{\mathbf{b}}^k, \mathbf{m}^k)$, one triplet for each layer. The tilde symbols denote that the weights and biases may have been modified through unrolling, absorption of batch-normalization parameters, etc. The masks may be initialized with a random vector or may be directly obtained from the first sample of the sequence. These triplets will be used during inference to compute the contracted $(Q, \mathbf{q})$ parameters.

During inference, we iterate through the sequence sample by sample, and compute the network output following (5.8). For each sample, we also check whether the reference point $\mathbf{x}_*$ should be reset using one of the criteria in Sec. 5.2.2. The following pseudo-code summarizes the workflow:

**Input:** Feed-forward network $F(\mathbf{x})$; sequential dataset $\mathcal{X}$.
**Output:** Contracted network $C_F(\mathbf{x})$; network output $\mathcal{Y}$.
Transform each layer into triplet $(\tilde{W}^k, \tilde{\mathbf{b}}^k, \mathbf{m}^k)$ (Sec. 5.2.1).
**for all** $\mathbf{x}_t \in \mathcal{X}$ **do**
  Compute mask update criterion (Sec. 5.2.2).
  **if** mask update criterion $>$ threshold **then**
    • Reset reference point $\mathbf{x}_* = \mathbf{x}_t$.
    • Do regular forward pass to update masks $\mathbf{m}^k$ and obtain network output.
    • Contract model by computing $(Q, \mathbf{q})$.
  **else**
    Compute output of contracted network (5.8).
  **end if**
**end for**

### 5.2.4 Computational Cost

The inference cost of a contracted network is compared against that of a standard multi-layer network. In a contracted network, the cost comes from the multiplication of the (flattened) input samples with the $Q$ matrix and addition of the bias $\mathbf{q}$ (and possibly the application of a final activation function on the resulting output values). Assuming a $Q$ matrix of dimension $(N_0 \times N_n)$ (where $N_0$ is the number of neurons in the first masked layer and $N_n$ the number of neurons in the final masked layer), and a bias vector $\mathbf{q}$ of $N_n$ elements, the number of computations needed for inference is

$$\mathcal{C}_{\text{contracted}} = 2N_0N_n + N_n = (2N_0 + 1)N_n. \tag{5.12}$$

In contrast, the number of computations for the original network is proportional to the number of neurons in each layer times their fan-in $f_{\text{in}}$ (i. e. the number of incoming connections):

$$\mathcal{C}_{\text{standard}} = \sum_{k}^{n} 2N_k f_{\text{in},k} + N_k = \sum_{k}^{n} (2f_{\text{in},k} + 1)\, N_k. \tag{5.13}$$

The inference cost in the contracted model only depends on the sizes of the input and output, whereas the inference cost in a multi-layer ANN scales with the number of layers and its internal connectivity. This fact becomes increasingly important in networks with hundreds of layers such as Inception and ResNet. On the other hand, the proportionality factors $N_0$ and $N_n$ in (5.12) may become prohibitively large in tasks like object detection or scene segmentation, where multi-channel heat maps are produced as output instead of the output of a limited number of classification neurons.

Having determined the inference cost of individual samples, we now factor in the mask updates to derive the total inference cost of the contracted network. To obtain the activation pattern needed for the masks, one can simply perform a standard forward pass in the original network. Another forward pass with the contracted network is not necessary for the current sample because the output of the original network can be used directly. The newly obtained masks allow updating of the $(Q, \mathbf{q})$ parameters, which are used for inference with the subsequent samples. In a dedicated hardware implementation, this computation of the contracted parameters can occur in the background, using spare compute cycles, even on the host controller on a separate idle thread. That way, the updates of the masks occur concurrently with the low-latency forward passes for inference and do not impede latency. We therefore consider the cost of inference in the contracted model at the time of an update to be that of a forward pass in the original network, $\mathcal{C}_{\text{standard}}$.

The cost of computing a mask update indicator (like *MSE input*) in the worst case consists of computing the MSE on two feature maps, which is negligible compared to the inference cost.

Let $r_{\text{update}} = \#\text{updates}/\#\text{samples}$ denote the mask update rate and

$$r_{\mathcal{C}} = \frac{\mathcal{C}_{\text{no-update}}}{\mathcal{C}_{\text{update}}} = \frac{\mathcal{C}_{\text{contracted}}}{\mathcal{C}_{\text{standard}}} \tag{5.14}$$

the cost ratio of a forward pass without and with mask updates. The compression rate is then given by

$$r = r_{\mathcal{C}} + r_{\text{update}}(1 - r_{\mathcal{C}}). \tag{5.15}$$

In the limit where the contracted network cost is much smaller than the cost of the original network ($r_{\mathcal{C}} \ll 1$), the compression rate is simply given by the update rate: $r \approx r_{\text{update}}$. There are two limit cases in which the network contraction method brings little to no gain, i. e. $r \approx 1$. The first case occurs if the masks need to be

updated for almost every sample, i. e. $r_{\text{update}} \approx 1$; the second case if the cost of the contracted network is about the same as the original network, i. e. $r_{\mathcal{C}} \approx 1$.

As a first example, consider LeNet-5 (LeCun *et al.*, 1998), a 4-layer convolutional neural network with 1.2 million parameters, which receives as input gray-scale images of dimension $28 \times 28 \times 1$. The labelled output of the network falls into one of 10 classes. Then, $Q$ is a $(10 \times 784)$ matrix, and $\mathbf{q}$ is a bias vector of size 10. In one forward pass, $\mathcal{C}_{\text{contracted}} = 15.6$ kOp and $\mathcal{C}_{\text{standard}} = 2340$ kOp for each frame.

As second example, the Inception-V3 architecture has an input dimension of $299 \times 299 \times 3$ and 1000 output classes, resulting in an inference cost of 0.53 G and 11.4 G operations for the contracted and original network, respectively.

Assuming that the masks must be recomputed every second frame ($1/r_{\text{update}} = 2$), the contracted LeNet-5 and Inception-V3 architectures would see a reduction in computations by $1/r = 2\times$ and $1.92\times$, respectively. With the $Q$ matrix updated every 10th frame, the computes are reduced by $1/r = 9.1\times$ and $7.69\times$, respectively.

Table 5.1 lists these compute costs for the tasks discussed in Sec. 5.3.

## 5.3 RESULTS

We implemented the network contraction method in Python using Keras with the TensorFlow backend. The complete code is available on GitHub[3].

We chose a variety of natural video datasets to evaluate the usefulness of this method in different settings, described in the next subsections. Most of the videos contain rich temporal and spatial dynamics, which serves to demonstrate that the mask update mechanism does not miss critical changes but makes optimal use of static episodes. In some datasets, ground-truth is available so we can explicitly measure how the model accuracy is affected by the masking and contraction. Where no ground truth is present, we use the output of the original network as reference for the masked model. Example videos from our results are available online to illustrate the performance of the contracted networks, compare them to the original networks and, where applicable, to the ground-truth. The summarized results in Table 5.1 will be discussed with each subsection describing a particular dataset.

### 5.3.1 Car Detection

*Dataset:* We use a 50 s video recording of highway driving from a Udacity online course on autonomous driving[4].

*Model:* The task is to put bounding boxes around cars on the highway. We first trained a 5-layer CNN to do binary classification (car / no car) on thumbnail images of cars or background. These learned car detector features are then swept across each frame of the highway driving clip, producing a heatmap of the class “car". This heatmap is then thresholded and the remaining blobs are covered with bounding boxes.

3 Link will be published on acceptance of the paper.
4 https://github.com/udacity/CarND-Vehicle-Detection

*Contraction:* The 5-layer CNN is contracted into one $(Q, \mathbf{q})$ layer, reducing the number of operations by 80×.

*Mask updates:* To evaluate how well the contracted model performs the car tracking task with fewer mask updates, we varied the threshold hyperparameter for three of the update criteria[5] and measured the MSE between the contracted and the original model (Fig. 5.4). The error increases linearly in this range, with the static update at every nth frame following the original model most accurately. After closer inspection, we find that *MSE input* sometimes triggers updates due to changes in the ambient scene rather than the relatively small shifts of other cars on the road. An illustration is shown in Fig. 5.3, where the appearance of large trees, shadows, or a changed paving triggers mask updates even though the small relative motion of the cars would not require it.

Using *MSE input* with mask updates at every 14th frame on average results in a compression of 12.3× in terms of computations. The associated output video[6] compares the car tracking performance of the contracted (red bounding boxes) and the original network (green boxes).

### 5.3.2 Pose Estimation on Surveillance Data

*Dataset:* The videos in the CAVIAR surveillance dataset[7] were recorded with a wide angle lens along the hallway of a shopping center in Lisbon. The video frames have a resolution of 384 × 288 pixels and are recorded at 25 fps. We use the same seven test videos as related work (Cavigelli *et al.*, 2020).

*Model:* We use the OpenPose network for a pose estimation task (Cao *et al.*, 2017). The architecture consists of a VGG-like trunk of 12 convolution layers and 3 pooling layers. The network then splits up into two branches, one for identifying body parts, the other for connecting them appropriately. Both branches consist of 5 convolution layers followed by 5 stages of 7 convolution layers each, for a total of 40 convolution layers per branch.

*Contraction:* We apply feature masking on both branches, but not on the trunk, for reasons that will become apparent in the next paragraph. The resulting architecture consists of the trunk plus two fully-connected output layers replacing the two 40-layer branches. The two branches of the original network require 155 billion operations in total, which are reduced to 44 billion operations after contraction. Together with the 61 billion operations for the trunk, the total reduction in computations is 2.06×.

*Mask updates:* To quantify which layers are likely to require few mask updates, we count the number of neurons for each layer that switch activation sign between consecutive frames, and average this number across all video sequences of the test set. The result is shown in Fig. 5.5. We observe a trend as described in Balduzzi *et al.*, 2017, where *contiguity* (the sequence length of activations with the same sign) increases with network depth. In other words, neurons in higher layers (past the

5 Since the whole network was contracted, the *number of activation sign changes* was not applicable here.
6 https://youtu.be/QYSa0r1DI00
7 CAVIAR was funded by the EC project IST 2001 37540 and is available online (http://homepages.inf.ed.ac.uk/rbf/CAVIAR/)

12th layer) tend to maintain the same sign from frame to frame. This observation is in line with the common observation that higher layers which encode more high-level concepts are expected to be stable across longer time periods. Further support was offered recently by Jastrzebski *et al.*, 2018, who applied the first-order Taylor expansion to each residual block of a ResNet architecture and showed that while lower layers learn new representations, higher layers tend to iteratively refine the hierarchical representations of lower layers. This, together with the 40% fewer activation sign changes in the upper two branches of the present architecture, motivates our choice to contract these layers but not the VGG trunk.

It is advantageous for the network contraction when the activation sign changes are correlated between layers, because this allows the mask update step to be done synchronously across all layers. For each video sequence in the test set, we compute the correlation between activation sign changes in the first layer versus all other layers and obtain a median correlation value of 0.95. This result indicates that a mask update trigger is appropriate for updating all masks simultaneously.

To quantify the performance of the contracted network, we measured the MSE between the output of the masked network and the original OpenPose network, while gradually increasing the threshold for mask updates, thereby trading off accuracy vs computational cost. Fig. 5.7 depicts this trade-off, and compares the performance of each of the four proposed update indicators. Overall, the MSE increases only moderately with decreasing update rates, with little dependence on the update mechanism. In contrast to the car tracking task (Sec. 5.3.1), updates based on dynamic indicators lead to slightly increased performance than updates at regular intervals. The improved behavior of the *MSE input* metric is not surprising: With static indoor surveillance videos, *MSE input* is less likely to be distracted by task-unrelated changes in the scene (see Fig. 5.6 for an illustration).

For the two sequences that gave the best and the worst results, we provide online videos that show the pose estimation results at different update rates[8]. The reduction in computations and the relative increase in error are listed in Table 5.1 for the lowest performing sequence.

### 5.3.3 Video Object Detection

*Datasets:* The two datasets are the KITTI dataset and ImageNet object-detection-from-video challenge. The object tracking test set of the KITTI Vision Benchmark Suite (Geiger *et al.*, 2012) consists of 29 videos of urban driving scenes; typical objects to be tracked include cars, buses, pedestrians, bicycles, traffic lights and street signs. The labels provided for the ImageNet dataset do not all match the classes of the COCO dataset with which our model was trained, so we selected a subset of ImageNet videos where the labels matched the known classes. Fig. 5.8 illustrates the masking method applied on two randomly selected clips from the two datasets.

*Model:* We use the YOLO architecture (Redmon *et al.*, 2016) pretrained on the COCO dataset[9]. The network shown in Fig. 5.9, consists of a trunk of 12 convolution layers (interleaved with batch normalization and pooling). The graph then splits

8 https://www.youtube.com/playlist?list=PLUK3dDFZv51JlzP_SH41h7Fws-aefyuCK
9 http://cocodataset.org

into a branch with 7 convolution layers and a skip-connection branch with a single convolution layer. The two branches merge to pass through two more convolution layers, the second of which contains separate channels for the object bounding boxes, the confidence scores, and the 80 different class predictions.

*Contraction:* The feature masking method is applied to the upper block of 7 convolution layers (red box in Fig. 5.9). A challenge in applying the network contraction method to this architecture is that both the input and the output dimensions of the masked section are large (c. f. Fig. 5.9), resulting in a large $Q$ matrix (6 billion entries). The size of the $Q$ matrix is the reason why the contraction method does not appear to bring a computational advantage over the original model: The matrix-vector product of the input with the $Q$ matrix during inference costs just as much as the forward pass through the 7 uncontracted convolution layers, namely 12 billion operations. Albeit the YOLO architecture may not be suited for our network contraction approach, this experiment is nevertheless worthwhile for showing that our dynamic feature masking can be applied successfully to the task of multiple object tracking.

*Mask updates:* As before, we analyze the performance trade-off by measuring the mAP while varying the update rate, using our proposed update indicators. For the *MSE input* metric we take the input to the contracted block of the network. For the *number of activation sign changes*, we use the activations of the first layer of this block as indicated by a star in Fig. 5.9. The *MSE output* criterion was not applicable here because of the multi-functional structure of the output layer.

The contracted model is more robust to low update rates in static scenes (recorded from a stationary car, Fig. 5.10 bottom) than in dynamic scenes (taken from a moving car, Fig. 5.10 top). In both situations, updates triggered by *activation sign changes* achieve highest mAP, regular update rates achieve lowest.

Annotated videos of both the original and our masked model are provided for visual comparison[10].

For the ImageNet video dataset, the performance trade-off (Fig. 5.11) is qualitatively similar to the KITTI task, with the update metrics ranking in the same order. The similar performance of the *MSE input* and *activation sign changes* metric is supported by the observation that the Pearson correlation coefficients between the two quantities lie above 0.9 for each of the masked layers. On average, the mAP remains within 10% of the precision of the original model for update rates beyond 40 frames.

### 5.3.4 Denoising Autoencoder

*Dataset:* The temporal MNIST (t-MNIST) dataset is a similarity-ordered version of the MNIST handwritten-digit dataset (LeCun *et al.*, 1998).

*Model:* The task is to denoise images from this dataset with added Gaussian noise (Fig. 5.12). The network is a DAE where the encoder consists of two convolution layers followed by a fully-connected layer, mapping the $28 \times 28$ gray-scale input images into a 16-dimensional latent space. The decoder section consists of

10 https://www.youtube.com/playlist?list=PLUK3dDFZv51KGIT5h1iJLp__JxtJ7BKn-

a fully-connected layer and three transpose-convolution layers, which map the 16-dimensional encoded representation back into image space.

*Contraction:* Only the decoder is contracted to benefit from the low-dimensional latent representation. As in the other experiments (see e. g. Fig. 5.5), the number of activation sign changes tends to decrease from lower to higher layers, which further motivates masking upper parts of the network. The resulting architecture consist of two convolution layers and two fully-connected layers, the last being the contracted decoder.

In general, autoencoders are advantageous for network contraction because the latent representation of the DAE architecture is low-dimensional, resulting in 2.2× fewer parameters, 4.1× fewer neurons, and 22.7× fewer operations for this architecture (c. f. Table 5.1), while maintaining good denoising performance[11].

*Mask updates:* The update indicator based on *number of activation sign changes* could not be used in this experiment because the latent representation (which feeds into the contracted part) does not use a ReLU function. The *MSE input* metric appears to be a good predictor of when to update the masks, as indicated by a high correlation between the *MSE input* and the number of activation sign changes in the two transposed convolution layers of the decoder. The dynamic criteria (*MSE input / output* in Fig. 5.13) perform consistently better than a fixed update at every $n$-th frame, which is in line with previous observations (see e. g. Figures 5.7, 5.10, and 5.11).

*Taylor approximation:* The first-order Taylor expansion of the original network behaves just as the contracted model (cf. Fig. 5.13), which supports the relation of our method with Taylor approximations derived in Sec. 5.2.1.

### 5.3.5 Retraining with ReLU Variant

The method of contracting several layers of a network into a single $Q$ matrix using dynamically updating activation masks can be applied to a pre-trained feed-forward network without retraining. However, one may consider retraining the network to reduce the number of mask updates needed during inference.

We consider the use of leaky ReLU activation functions. An inaccurate mask introduces errors during inference by blocking some positive activations or transmitting some negative activations. These errors are most notable if the masks are used to replace non-leaky ReLUs. For leaky ReLUs, the error due to an outdated mask entry gets smaller, the higher the leak $\alpha$. This fact can be used during training, by gradually increasing the slope of the left branch of the ReLU. This process progressively linearizes the network and is thus strictly limited by a leak $\alpha < 1$. As $\alpha$ approaches 1, the network becomes insensitive to errors in the masks, but also loses expressive power. On the other hand, linear networks do not suffer from the "shattered gradients" problem, which is why Balduzzi *et al.*, 2017 initialize their networks to "look linear" at first, with the symmetry in the activation function gradually breaking during training.

To explore the effect of leaky ReLUs on the masking, we train a DAE on the t-MNIST dataset while increasing the leak parameter $\alpha$ at certain epochs, which

11 https://youtu.be/bBFTkyckQe8

are evident by the kinks in the training curve (Fig. 5.14). The model classification performance degrades only little when increasing the linearity of the rectifier. A similar convergence behavior is observed when training a 6-layer CNN and a 32-layer ResNet on CIFAR-10 (not shown).

After training the decoder of the DAE with increasing linearity, we test our model contraction method on the almost linear model ($\alpha = 0.89$). The result shown in Fig. 5.13 confirms our initial hypothesis: The contracted leaky ReLU model shows a consistently lower error compared to the contracted ReLU model. The almost linear rectifiers make the model robust to outdated masks.

## 5.4 DISCUSSION

The results from the experiments on the different datasets show that the contracted networks maintain good accuracy which is corroborated in the case of the t-MNIST and ImageNet videos where ground truth is available. This finding demonstrates that deep neural networks can be approximated as a first-order Taylor expansion with respect to the input around a dynamically updating reference point (c. f. Sec. 5.2.1).

We found that the contracted networks lead to a compression factor of $1 - 12\times$ depending on their architecture. The best architectures for contraction have either a low-dimensional input or output, or an intermediate bottleneck layer, as in the autoencoder for denoising t-MNIST. *Contractive autoencoders* (Rifai *et al.*, 2011) offer additional benefits because their loss function contains a term proportional to the Frobenius norm of the encoder's Jacobian. This penalizer encourages the mapping to the latent space to be contractive in the neighborhood of the training data, making the encoded representation robust to small changes of the input. We hypothesize that a contracted decoder that receives these stabilized latent representations as input will have to update its masks less frequently.

Mask updates can be performed at regular intervals or triggered dynamically based on a specific indicator metric. The indicator based on the *number of activation sign changes* is most closely correlated with outdated masks, and cheaper to compute than using the *MSE input / output* metrics. This indicator also requires the least mask updates to maintain model accuracy on the pose-estimation and object detection tasks on KITTI and the ImageNet video datasets. This indicator was not applicable in the car tracking and denoising tasks, either because the indicator layer has no ReLU or the entire network was contracted. An alternative is the data-driven predictor *MSE input*, which achieved better model accuracy than regular updates at every n-th frame with one exception: The car tracking example showed that the *input MSE* may be biased by changes in the ambient scene, which makes updates at regular intervals a safer choice whenever the *activation sign changes* indicator is not applicable.

## 5.5 CONCLUSION

We present a network approximation scheme that replaces the layer nonlinearities by dynamically updated masks, and show its equivalence to a first-order Taylor expansion through theory and experiments. Among the four mask indicators, our results show that the optimal indicator is based on the count of activation sign changes, is cheap to compute and requires fewer updates at iso-accuracy than the other indicators. By compressing a network using this masking technique, the computational cost of processing a stack of layers is reduced to the cost of performing a single matrix-vector multiplication.

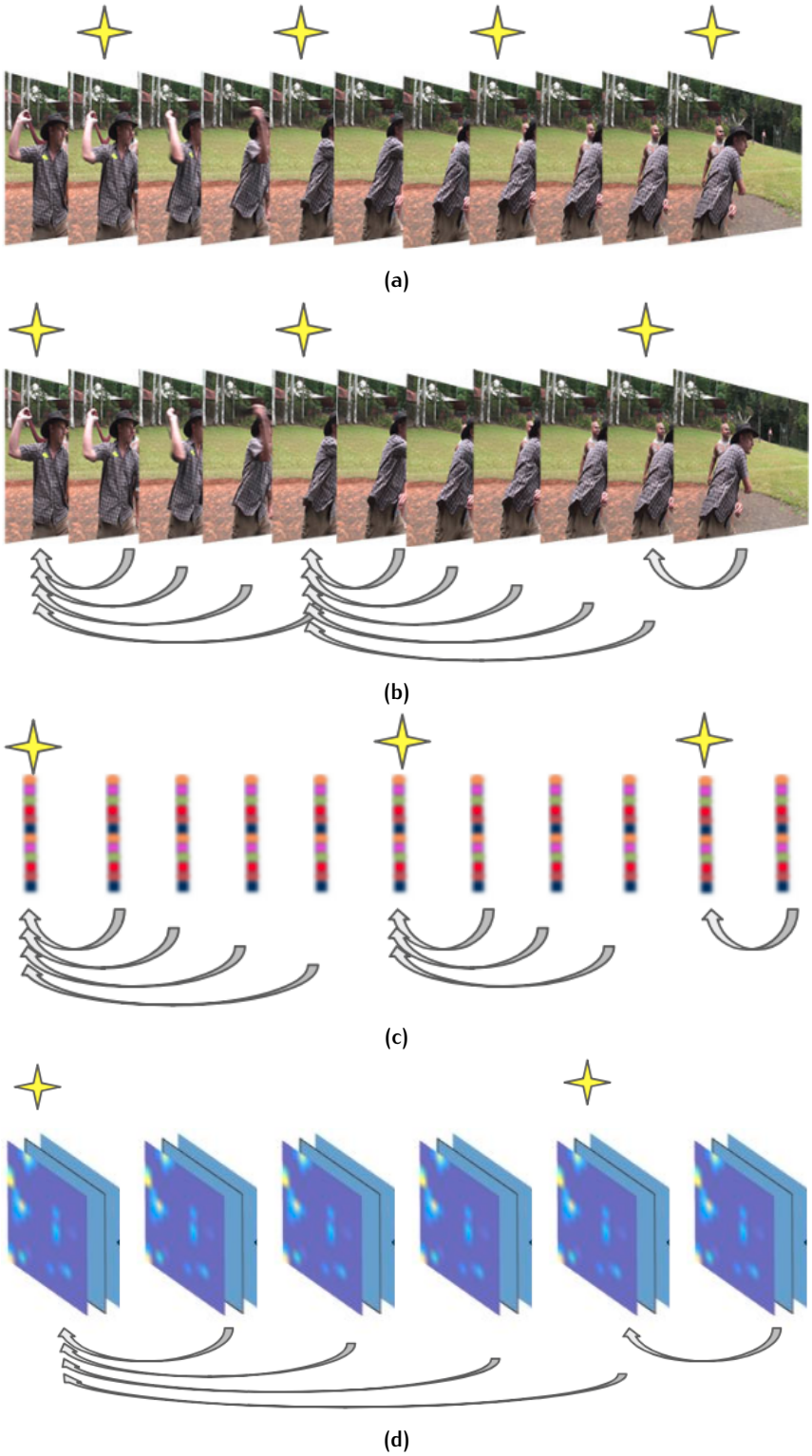


**Figure 5.2:** Illustration of mask update indicators. The stars denote when masks are updated. The arrows mark how successive frames are compared against the frame of the previous update. 5.2a: Regular update. 5.2b: Update when MSE on input images exceeds threshold. 5.2c: Update when MSE on output activations exceeds threshold. 5.2d: Update when the number of activation sign changes exceeds threshold. See Sec. 5.2.2 for details.

**Table 5.1:** Summary of experiments using different architectures.

| Dataset | | Udacity | CAVIAR surveillance | KITTI | ImageNet video | t-MNIST |
|---|---|---|---|---|---|---|
| Task | | Tracking | Pose estimation | Object detection | Object detection | Denoising |
| Architecture | original | 5 Conv | 90 Conv | 23 Conv | 23 Conv | 5 Conv, 2 FC |
| | masked | 1 FC | 10 Conv, 2 FC | 16 Conv, 1 FC | 16 Conv, 1 FC | 2 Conv, 2 FC |
| Dimensions | input | $1280 \times 300 \times 3$ | $36 \times 48 \times 128$ | $13 \times 13 \times 512$ | $13 \times 13 \times 512$ | 16 |
| | output | $153 \times 30 \times 1$ | $36 \times 48 \times 57$ | $13 \times 13 \times 1024$ | $13 \times 13 \times 1024$ | $28 \times 28 \times 1$ |
| Metric | original | - | - | - | 91.51 mAP | 0.016 MSE |
| | masked | 0.013 MSE to orig. | 2.88e-4 MSE to orig. | - | 88.75 mAP | 0.023 MSE |
| Comput. reduction | $1/r_c$ | 80× | 2.1× | 1× | 1× | 22.7× |
| Mask updates | $1/r_{\text{update}}$ | 14.3 | 10.1 | 10 | 40 | 3 |
| Compression | $1/r$ | 12.3× | 1.9× | 1× | 1× | 2.8× |


**Figure 5.3:** Illustration of the car tracking task with bounding boxes from the original and contracted model.

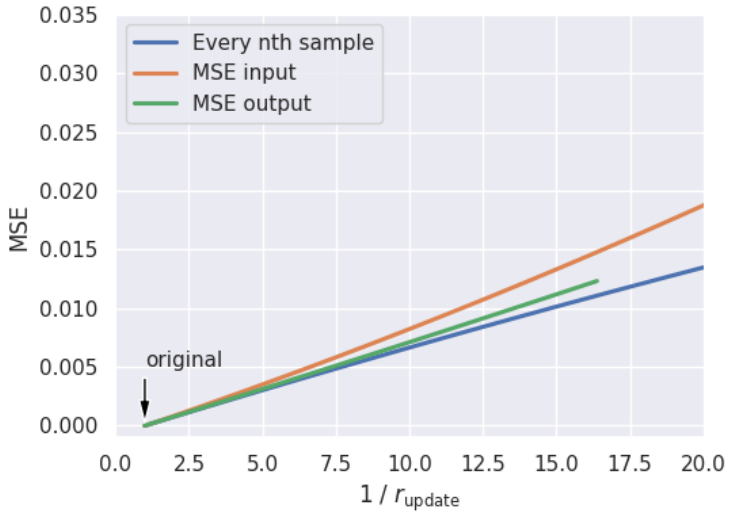


**Figure 5.4:** Mean square error between the contracted and original network as a function of the mask update rate, $r_{update}$, for the car tracking task.

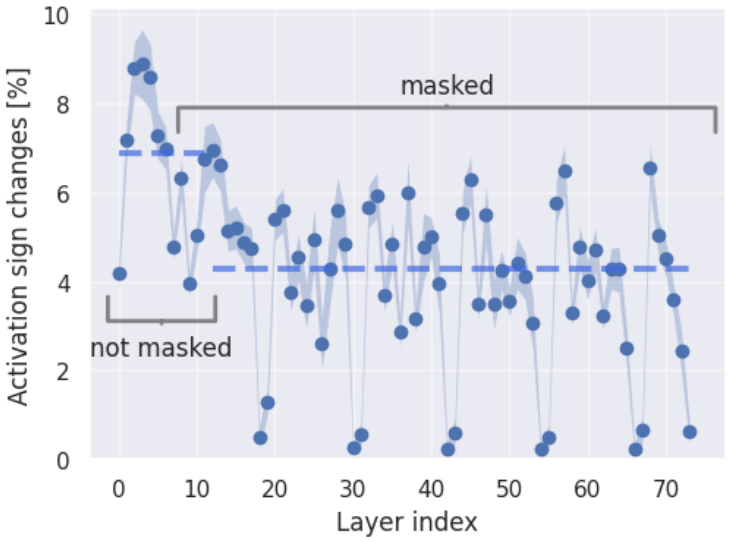


**Figure 5.5:** Percentage of neurons that change their activation sign value between consecutive frames. Numbers are averaged over test sequences of the pose estimation task. The shaded region denotes the 3 times the standard deviation. The first 12 convolution layers are not masked, the remaining 68 are masked. Dashed lines represent the median number of activation sign changes for both the masked and non-masked layers. The median value for masked layers is 40% lower than that of the masked layers. The low peaks correspond to the $1 \times 1$ convolution layers.

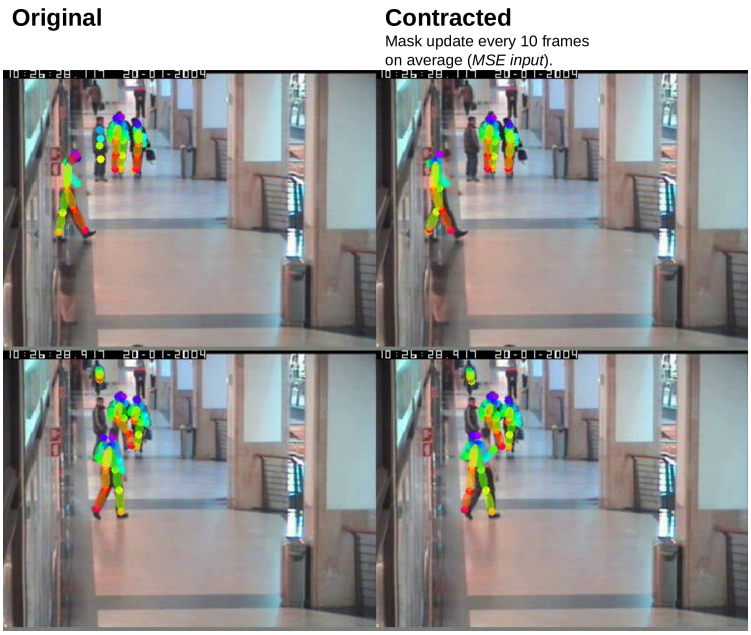


**Figure 5.6:** Example comparison of the pose estimation of the original and the contracted model. Colors indicate the different body limbs.

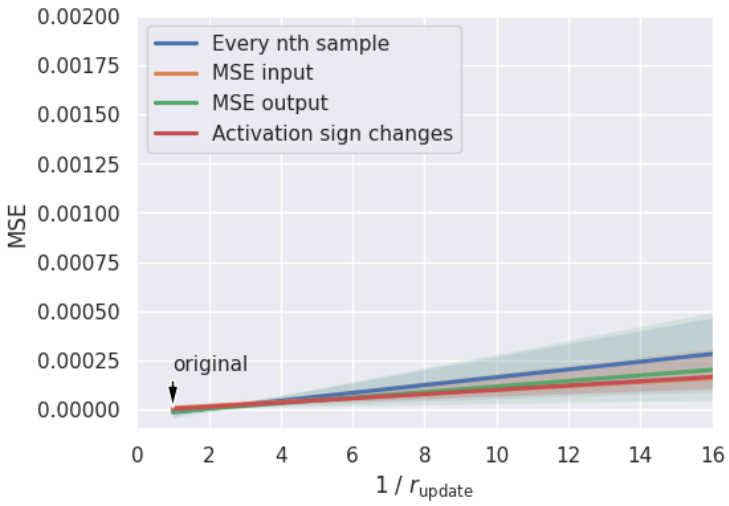


**Figure 5.7:** Mean square error between the contracted and original network as a function of the mask update rate, $r_{update}$, for the pose estimation task. The shaded region denotes the 3-sigma confidence interval of the regression line for the 7 test sequences.


**Figure 5.8:** Illustrative comparison of the original and the contracted model output for the object detection task using ImageNet (left) and KITTI (right).

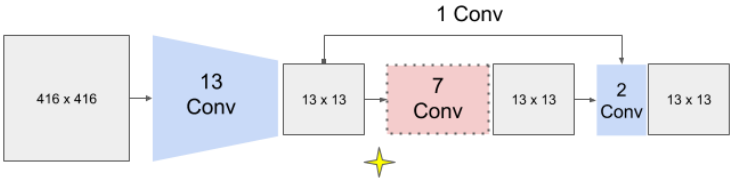


**Figure 5.9:** YOLO architecture used for object detection in the KITTI and ImageNet video experiments. The upper 7 convolution layers (red, dotted border) are masked. Sign changes in activations of the first masked layer predict mask updates (yellow star). The gray boxes denote the feature map size of the final layer of a convolution block.

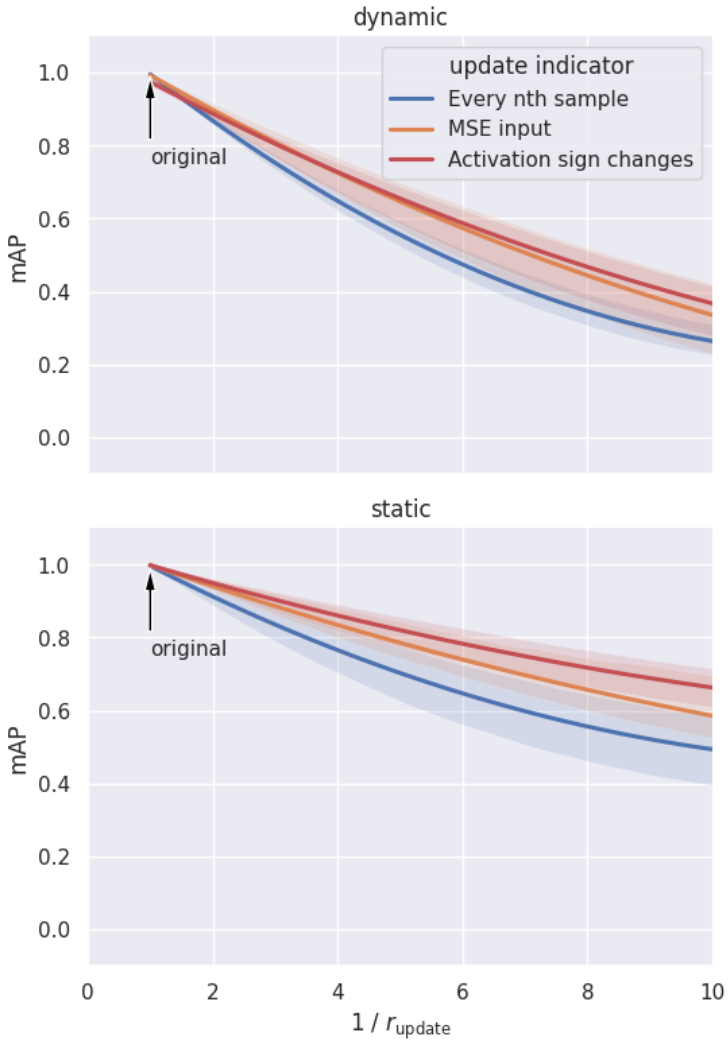


**Figure 5.10:** mean Average Precision (mAP) plotted against the mask update rate for the KITTI object detection task. Each line is obtained from a linear regression of the mAP values across the test samples. The shaded region denotes the 3-sigma confidence interval. We used the labels and bounding boxes predicted by the original model as ground truth for the contracted model. The two panels separately show videos taken from a moving car (20 clips, top) and stationary car (9 clips, bottom).

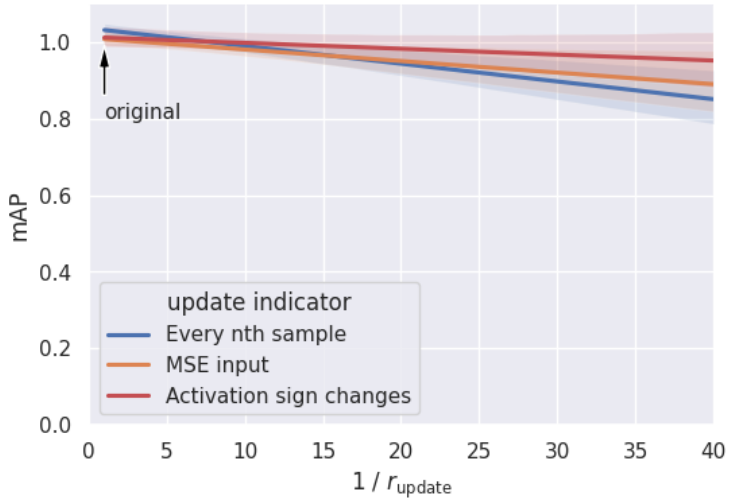


**Figure 5.11:** Least-squares fit of the mean Average Precision for the ImageNet video task. The reported mAP values of the contracted model were normalized by the mAP of the original model to facilitate the regression.


**Figure 5.12:** Illustration of the denoising autoencoder task, with the noisy model input (top left), training target (top right), denoised output of the original model (bottom right), and output of the contracted model (bottom left).

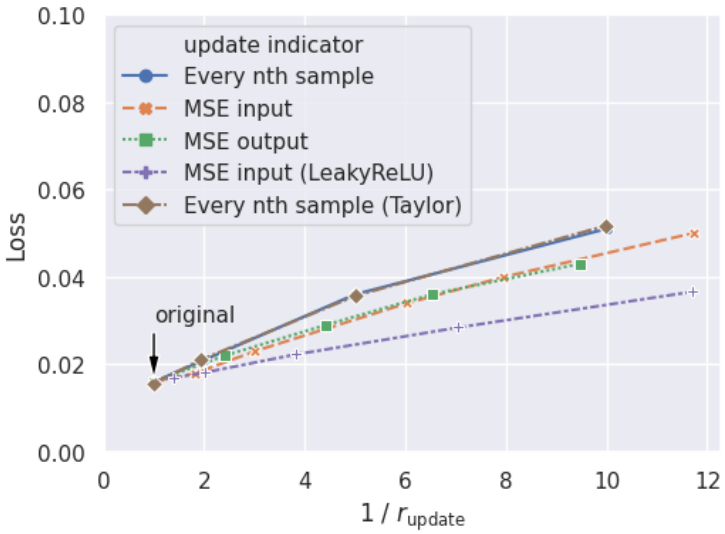


**Figure 5.13:** Loss (reconstruction MSE) plotted against the mask update rate for the Denoising AutoEncoder (DAE) task on t-MNIST.

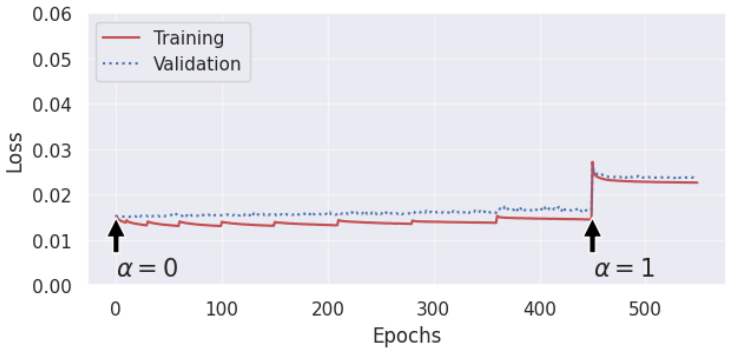


**Figure 5.14:** Accuracy during refinement of the pre-trained DAE on the t-MNIST. During training, the slope parameter $\alpha$ of the Leaky ReLU activation function is increased from $\alpha = 0$ at epoch 0 to $\alpha = 1$ at epoch 520.

# 6 CONCLUSION

In this thesis we studied bio-inspired sensors, processors and algorithms for computer vision, with the aim to create artificial systems that inherit some of the computational power and energy efficiency of the brain. As in any such endeavor, more questions were raised than answered. Here we summarize some of the lessons learnt, and outline future work.

## 6.1 SUMMARY OF CONTRIBUTION

The main contribution of this thesis concerns the advancement of techniques to convert ANNs into SNNs, with the aim to obtain efficient models that are compatible with event-based sensor input and asynchronous processing on neuromorphic hardware. We approached this objective from three directions, covering the algorithms, the processor, and the sensor.

**ALGORITHMS.** We explored three spike encoding schemes for SNNs, based on the mean firing rate, spike latency, and spike patterns. We showed mathematically that a rate code approaches the original ANN asymptotically, and validated the result in experiments on state-of-the-art models for ImageNet. To address the issue of increased operational cost of rate codes, we derived a latency code with minimal spike count, which was validated on MNIST. To overcome the lack of robustness to noise inherent in the latency code, we developed an encoding scheme that utilizes a sequence of spikes, each of which carries information in their timing. This last code achieved the best results overall, scaling all the way up to ImageNet while reducing the operational cost significantly compared to the rate-coded SNN.

Aside from these algorithmic advances, a more technical contribution to the community concerned the development of a toolbox for SNNs that automates the process of mapping a pre-trained ANN into the spike domain, and running it on various SNN simulators or dedicated hardware platforms.

**PROCESSING.** While most of the algorithmic work had been carried out in simulation, we were able to deploy some of our rate-coded models on the Loihi neuromorphic chip, and benchmarked them on several computer vision tasks. As with the SNN toolbox, a practical result of this work with Loihi was the development of a DNN compiler that greatly facilitates the process of deploying large networks on Loihi. We hope that this tool will lead to a shift from simulation-based SNN characterization towards a more reliable performance profiling of SNN models on neuromorphic hardware.

**SENSING.** We demonstrated the benefits of using event-based sensors like the DVS in conjunction with event-based algorithms, e. g. for optical flow and vision processing in SNNs. In a short excursion to neurophysiology we showed that the DVS can be used to drive RGCs of mice in vitro, and characterized the stimulus properties that elicit optimal cell response. Again, the toolchain to drive an MEA from the DVS is a practical contribution which lays the foundation for a more rigorous exploration of this approach in the context of retinal prostheses.

## 6.2 LIMITATIONS AND OUTLOOK

**RATE-BASED CONVERSION.** Building on an outstanding body of prior work, we began this journey by scaling up the early methods of ANN-to-SNN conversion from MNIST all the way to ImageNet (Sec. 2.1). Soon we had to admit that conclusions drawn from MNIST offer no guarantees whatsoever that a method is able to solve more difficult tasks based on natural images. Unfortunately, some studies continue to place disproportionate weight on this dataset. Another realization when dealing with SNNs was that, in contrast to ANNs, the accuracy cannot be simply increased by adding layers and throwing more data at the model. Quite the opposite - the deeper a spiking networks gets, the more chance it has to accumulate discretization errors across the hierarchy. Regularizing or clamping higher layers as done in Sec. 2.1.3.3 provides partial remedy. A more elegant solution might be found in residual architectures (Hu *et al.*, 2018; Kugele *et al.*, 2020), though the exact effect of skip-connections on spike rates in higher layers remains to be understood.

**OPERATIONAL COST.** Another painful realization from our work with SNNs was that the regularly repeated claim of superior efficiency seemed unlikely to hold in case of rate-coded deep models. Our converted GoogleNet architecture actually required significantly *more* operations to reach the accuracy of the corresponding ANN. True, these operations are additions and thus cheaper than the MACs in ANNs. But - and this is another limitation of most algorithmic SNN studies - the pure operation count neglects to factor in the cost of asynchronous memory accesses in state-full, spike-based models. Reporting realistic estimates of power consumption and run-time would greatly help gauge the actual value of the many new SNN conversion and training methods. That of course would require detailed models of the hardware substrate or tedious implementation on neuromorphic platforms.

**TEMPORAL CODES.** Along with others, we took the redundancy in rate codes as an incentive to study alternative spike codes. Our single-spike TTFS approach (Sec. 2.2) maximally reduces spike count, but suffers from the same limitation addressed earlier: In its current form it does not go beyond MNIST. We demonstrated many partial improvements, but a complete solution within this coding scheme would require a layer-wise processing as in the ANN. We showed in (Sec. 2.3) that, if one is willing to adopt layer-wise processing, ANN-like accuracy can be achieved using a pattern code with significantly fewer spikes than in a rate-coded SNN. These results are promising, but the wide space of possible encodings is still largely unexplored.

For instance, one could envision a hybrid temporal code that extends TTFS by additional spikes that refine the initial signal. Though not strictly a temporal code, a promising method in this direction based on adaptation is proposed by Zambrano *et al.*, 2019.

**NEUROMORPHIC HARDWARE.** In a collaboration with Intel we had the opportunity to develop a DNN compiler for their neuromorphic chip Loihi. Already during the conversion work we aimed to make deep SNNs easily accessible, which resulted in the now widely used SNN toolbox. This motivation also guided the design of the Loihi compiler, which provides an interface derived from Tensorflow / Keras, thus softening the culture shock a DL researcher might experience when switching to neuromorphic hardware. Using this new tool we were able to perform a power- and latency-analysis of the MobileNet architecture for CIFAR-10 as well as SLAYER-trained models for neuromorphic datasets running on Loihi.

**EVENT-BASED SENSING AND PROCESSING.** As part of our experiments with event-based vision sensors, we investigated the possibility of injecting DVS events directly into converted SNNs. Since direct spike-based training continues to be a challenge, a typical scenario was to convert ANNs that had been trained on frames synthesized from the event stream. However, when directly injecting DVS events into these converted SNNs at inference time, we noticed a decrease in accuracy that could be explained by the temporal dynamics of the event stream, to which the frame-trained SNN was not accustomed. On the other hand, we were able to show a decrease in operational cost due to the sparse nature of the event stream, which demonstrates the potential benefit of combining event-based sensing and processing. Since then, several solutions have been proposed to obtain SNNs that are able to handle inhomogeneity in event-rates (c. f. Sec. 1.3). Not surprisingly, models that learn directly from events are particularly successful, though the high training cost leaves room for future developments. Research in this area will progress further as more truly dynamic (rather than synthesized) event-datasets like DVS-Gestures are made available. A highly interesting perspective might be gained then by combining properties of SNNs with those of gated LSTM units, which provide more flexible means of temporal integration.

**THE BLESSING OF HARDWARE CONSTRAINTS.** In standard ANNs, activation values are represented by 32-bit floating point numbers, which makes their dynamic range and numeric precision unlimited for all practical purposes. In contrast, both range and precision of activity values (e. g. spike rates) in the SNN are severely limited, either due to the hardware constraints, or the finite simulation duration and discrete time stepping of the simulation. For instance, an SNN run for 256 time steps may represent activation values up to 8 bit precision. In higher layers however, the available number of time steps is effectively decreased by the time it takes lower layers to ramp up their activity. In addition, the spike rates in these upper layers will be corrupted by noise passed on from lower layers. Thus, to minimize approximation errors in the converted SNN, the ANN will ideally have been trained with a limited band of quantized activation values, in anticipation of the SNN constraints.

Incidentally, the topic of quantized models is of great interest to the DL community e. g. for its implications on efficient inference in edge devices. In our view, the concept of reduced numeric precision should appear ubiquitously in SNN training and conversion, but is so far widely missing (for exceptions see e. g. Sorbaro *et al.*, 2020, also Sec. 2.3). As low-precision is enforced when deploying a model on neuromorphic hardware, we are presented here with a beautiful example of how a seeming restriction on the hardware side can evolve into a feature that is desirable from the algorithmic perspective, namely to reduce discretization errors. To take this argument one step further: With hardware constraints leading to improved algorithms, together they might also shed light on how neural computation is possible under the severe space and energy constraints present in the brain.

# A APPENDIX

## A.1 SUPPLEMENTARY MATERIAL TO SECTION 2.1

### A.1.1 Relation Between SNN Rates and ANN Activations

#### A.1.1.1 *Reset to Zero*

Our goal is to derive a relation between the firing rate $r_i^1(t)$ of a neuron $i$ in layer 1 of the SNN and the activation of the corresponding neuron in the ANN. To simplify the notation, we drop the layer and neuron indices. Starting from the membrane equation (2.4a), the average firing rate can simply be computed by summing over the simulation time $t$:

$$\sum_{t'=1}^{t} V(t') = \sum_{t'=1}^{t} \big(V(t'-1) + z\big)\,(1 - \Theta_{t'}) \tag{A.1}$$

Under the assumption of constant analog input $z$ to the first hidden layer (cf. Sec. 2.1.2.4), and using $N(t) := \sum_{t'=1}^{t} \Theta_{t'}$ we obtain

$$\sum_{t'=1}^{t} V(t') = \sum_{t'=1}^{t} V(t'-1)(1 - \Theta_{t'}) + z\left(\frac{t}{\Delta t} - N\right). \tag{A.2}$$

The time resolution $\Delta t$ enters the equation when evaluating the time-sum over a constant: $\sum_{t'}^{t} 1 = t/\Delta t$. It will be replaced by the definition of the maximum firing rate $r_{\max} = 1/\Delta t$ in the following.

By rearranging Equation (A.2) to yield the total number of spikes $N$, and dividing by the simulation time $t$ we obtain the average firing rate $r$ of a neuron in layer 1:

$$r := \frac{N}{t} = r_{\max} - \frac{1}{zt}\sum_{t'=1}^{t}(V(t') - V(t'-1)(1 - \Theta_{t'})). \tag{A.3}$$

By rearranging the indices in the sum on the right-hand side, equation (A.3) simplifies to

$$r = r_{\max} - \frac{V(t) - V(0)}{zt} - \frac{1}{zt}\sum_{t'=1}^{t} V(t'-1)\Theta_{t'}. \tag{A.4}$$

Since $\Theta_t$ equals 1 if there is a spike and 0 otherwise, the last term in equation (A.4) sums up the membrane potentials of the neuron just before a spike. In the case of reset-to-zero, constant input, constant threshold, and no leak, the value of the membrane potential immediately before a spike is always the same, and is always an integer multiple of the input $z$. Let therefore $n \in \mathbb{N}$ be the number of time-steps needed to cross threshold, i. e. $(n-1)z < V_{\text{thr}} \leq nz$. Then

$$\frac{1}{zt}\sum_{t'=1}^{t} V(t'-1)\Theta_{t'} = \frac{1}{zt}(n-1)zN = (n-1)r. \tag{A.5}$$

With this, the expression for the firing rate (A.3) becomes

$$r = \frac{1}{n}\left(r_{\max} - \frac{V(t) - V(0)}{zt}\right). \tag{A.6}$$

We now define the residual $\epsilon \in \mathbb{R}$ as the surplus charge above threshold at the time of spike:

$$\epsilon := nz - V_{\text{thr}}. \tag{A.7}$$

Plugging this into equation (A.6), the average spike rate is given by

$$r = \frac{z}{V_{\text{thr}} + \epsilon}\left(r_{\max} - \frac{V(t) - V(0)}{zt}\right). \tag{A.8}$$

Lastly, we make use of the fact that in the first hidden layer at constant input, $z^1 = V_{\text{thr}} a^1$. Setting $V(0) = 0$, reordering the terms, and reintroducing the dropped indices yields Equation (2.5a):

$$r_i^1(t) = a_i^1 r_{\max} \frac{V_{\text{thr}}}{V_{\text{thr}} + \epsilon_i^1} - \frac{V_i^1(t)}{t(V_{\text{thr}} + \epsilon_i^l)}. \tag{A.9}$$

#### A.1.1.2 *Reset by Subtraction*

The derivation of a relation between rates $r$ and activations $a$ simplifies greatly for the case of *reset by subtraction*. Averaging the membrane equation (2.4b) over the simulation time $t$ yields:

$$\frac{1}{t}\sum_{t'=1}^{t} V(t') = \frac{1}{t}\sum_{t'=1}^{t} V(t'-1) + zr_{\max} - V_{\text{thr}}\frac{N}{t}. \tag{A.10}$$

Using $z^1 = V_{\text{thr}} a^1$ and solving for $r = N/t$ leads to

$$r(t) = ar_{\max} - \frac{1}{V_{\text{thr}} t}\sum_{t'=1}^{t}(V(t') - V(t'-1)) = ar_{\max} - \frac{V(t) - V(0)}{V_{\text{thr}} t}. \tag{A.11}$$

Setting $V(0) = 0$ and reintroducing the indices yields Equation (2.5b).

## A.2 SUPPLEMENTARY MATERIAL TO SECTION 2.2

### A.2.1 Dynamical Systems Perspective

The dynamic range of biological neurons is limited. Similarly, it is beneficial in temporally encoded networks of artificial neurons to allow only a certain band of spike rates. An upper limit may be given by the refractory period or finite time resolution of the simulator. A lower limit occurs in neurons of *Class 2* (according to the classification after Hodgkin and Huxley). The transfer function of such neurons is zero for low inputs and shows a steep rise to nonzero frequencies at a certain input. The model and behavior of such neurons is best described in terms of dynamical systems theory.

Izhikevich identifies four generic bifurcation types. The first bifurcation, saddle-node on invariant cycle, occurs in *Class 1* neurons, i. e. monostable integrators. The other three bifurcations may undergo *Class 2* behavior. The subcritical as well as the supercritical Andronov-Hopf bifurcation occur in resonator neurons, which are not applicable to our current SNNs coding schemes. Thus we are left with the fourth, a saddle-node bifurcation, which occurs in bistable integrators, i. e. where a spiking limit cycle coexists with a saddle and a node.

This saddle-node bifurcation can be achieved with a one-dimensional system. A simple leaky IF with $V' = I - V$ is not sufficient because the temporal derivative of the membrane potential is monotonic. On the other hand, a quadratic IF neuron can show *Class 2* behavior if the reset potential is higher than the bifurcation point. When the input current ramps up and the system undergoes a bifurcation, the neuron can leave the stable point and enter the spiking limit cycle. Unfortunately, this behavior will not help in our temporal coding scheme because the time to first spike will be delayed by the "ghost" of the saddle-node (where $V'$ is close to zero).

The bifurcation analysis above relies on a rate-code, where we use only the first spike to recover our TTFS code. If instead we apply a TTFS code from the start, we can get *Class 2* excitability with the Izhikevich *Class 2* or *Phasic Spiking* model. This model requires a ramping PSP as input (a step PSP results in *Class 1* behavior). The leaky IF with TTFS code has the same transfer function. *Class 3* excitability (Izhikevich 7.1.4) should also produce the same result, with the difference that it needs a step inputs instead of ramps. The problem in all these cases is that the assumption of a constant input step size or ramp slope is not satisfied in our SNNs: Presynaptic neurons fire asynchronously and thereby change the input to a postsynaptic neuron over time, which will disturb its output spike time.

# PUBLICATIONS RESULTING FROM THIS WORK


[1] J. Ahn, B. Rueckauer, Y. Yoo, and Y. S. Goo, "New Features of Receptive Fields in Mouse Retina through Spike-triggered Covariance," *Experimental Neurobiology*, vol. 29, no. 1, pp. 38–49, 2020. DOI: 10.5607/en.2020.29.1.38. [Online]. Available: https://www.en-journal.org/journal/view.html?uid=508&vmd=Full.

[2] S. C. Liu, B. Rueckauer, E. Ceolini, A. Huber, and T. Delbruck, "Event-Driven Sensing for Efficient Perception: Vision and audition algorithms," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 29–37, 2019, ISSN: 15580792. DOI: 10.1109/MSP.2019.2928127.

[3] B. Rueckauer, C. Bybee, R. Goettsche, Y. Singh, J. Mishra, and A. Wild, "NxTF: A Compiler for Deep Spiking Neural Networks on Loihi," *Under review*, 2020.

[4] B. Rueckauer, N. Kaenzig, S.-C. Liu, T. Delbruck, and Y. Sandamirskaya, "Closing the Accuracy Gap in an Event-Based Visual Recognition Task," Tech. Rep., 2019. arXiv: 1906.08859. [Online]. Available: http://arxiv.org/abs/1906.08859.

[5] B. Rueckauer and S. C. Liu, "Linear approximation of deep neural networks for efficient inference on video data," in *European Signal Processing Conference*, 2019, ISBN: 9789082797039. DOI: 10.23919/EUSIPCO.2019.8902997.

[6] B. Rueckauer and S.-C. Liu, "Conversion of analog to spiking neural networks using sparse temporal coding," in *ISCAS*, Florence, Italy: IEEE, 2018, pp. 1–5, ISBN: 9781538648810. DOI: 10.1109/ISCAS.2018.8351295. [Online]. Available: https://ieeexplore.ieee.org/document/8351295/.

[7] B. Rueckauer and S.-C. Liu, "Contraction of dynamically masked deep neural networks for efficient video processing," *Under review*, 2020.

[8] B. Rueckauer and S.-C. Liu, "Temporal Pattern Coding in Deep Spiking Neural Networks," *Under review*, 2020.

[9] B. Rueckauer, I.-A. Lungu, Y. Hu, and M. Pfeiffer, "Theory and Tools for the Conversion of Analog to Spiking Convolutional Neural Networks," *NIPS workshop on computing with spikes*, 2016. arXiv: 1612.04052. [Online]. Available: http://arxiv.org/abs/1612.04052.

[10] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification," *Frontiers in Neuroscience*, vol. 11, no. December, pp. 1–12, 2017, ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00682. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnins.2017.00682/full.

# BIBLIOGRAPHY


[1] S. Afshar, T. J. Hamilton, J. Tapson, A. van Schaik, and G. Cohen, "Investigation of Event-Based Surfaces for High-Speed Detection, Unsupervised Feature Extraction, and Object Recognition," *Frontiers in Neuroscience*, vol. 12, no. January, pp. 1–19, 2019, ISSN: 1662-453X. DOI: 10.3389/fnins.2018.01047. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.01047/full.

[2] J. Ahn, M. H. Choi, K. Kim, S. S. Senok, D. I. Dan Cho, K. I. Koo, and Y. Goo, "The Advantage of Topographic Prominence-Adopted Filter for the Detection of Short-Latency Spikes of Retinal Ganglion Cells," *Korean Journal of Physiology and Pharmacology*, 2017, ISSN: 20933827. DOI: 10.4196/kjpp.2017.21.5.555.

[3] K. N. Ahn, J. Y. Ahn, J. H. Kim, K. Cho, K. I. Koo, S. S. Senok, and Y. S. Goo, "Effect of Stimulus Waveform of Biphasic Current Pulse on Retinal Ganglion Cell Responses in Retinal Degeneration (rd1) Mice," *Korean Journal of Physiology and Pharmacology*, 2015, ISSN: 20933827. DOI: 10.4196/kjpp.2015.19.2.167.

[4] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I. A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S. C. Liu, and T. Delbruck, "NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 3, pp. 644–656, 2019, ISSN: 21622388. DOI: 10.1109/TNNLS.2018.2852335.

[5] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. D. Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A Low Power, Fully Event-Based Gesture Recognition System," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7388–7397, ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.781. [Online]. Available: http://ieeexplore.ieee.org/document/8100264/.

[6] D. Balduzzi, M. Frean, L. Leary, J. Lewis, K. W.-D. Ma, and B. McWilliams, "The Shattered Gradients Problem: If Resnets Are the Answer, Then What Is the Question?" In *ICML*, 2017. [Online]. Available: http://arxiv.org/abs/1702.08591.

[7] D. Balduzzi, B. McWilliams, and T. Butler-Yeoman, "Neural Taylor Approximations: Convergence and Exploration in Rectifier Networks," in *ICML*, 2016. [Online]. Available: http://arxiv.org/abs/1611.02345.

[8] A. Basu, J. Acharya, T. Karnik, H. Liu, H. Li, J. S. Seo, and C. Song, "Low-Power, Adaptive Neuromorphic Systems: Recent Progress and Future Directions," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*,

vol. 8, no. 1, pp. 6–27, 2018, ISSN: 21563357. DOI: 10.1109/JETCAS.2018.2816339.

[9] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, and C. Eliasmith, "Nengo: A Python Tool for Building Large-Scale Functional Brain Models," *Frontiers in Neuroinformatics*, vol. 7, no. JAN, pp. 1–13, 2014, ISSN: 16625196. DOI: 10.3389/fninf.2013.00048.

[10] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long Short-Term Memory and Learning-to-Learn in Networks of Spiking Neurons," in *Advances in Neural Information Processing Systems*, 2018, pp. 787–797. [Online]. Available: http://arxiv.org/abs/1803.09574.

[11] E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup, "Conditional Computation in Neural Networks for Faster Models," in *International Conference on Learning Representations*, 2016, pp. 1–9. [Online]. Available: http://arxiv.org/abs/1511.06297.

[12] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014, ISSN: 00189219. DOI: 10.1109/JPROC.2014.2313565.

[13] R. Benosman, C. Clercq, X. Lagorce, S. H. Ieng, and C. Bartolozzi, "Event-Based Visual Flow," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 2, pp. 407–417, 2014, ISSN: 2162237X. DOI: 10.1109/TNNLS.2013.2273537.

[14] R. Berner, T. Delbruck, A. Civit-Balcells, and A. Linares-Barranco, "A 5 Meps $100 USB2.0 Address-Event Monitor-Sequencer Interface," in *Proceedings - IEEE International Symposium on Circuits and Systems*, 2007, pp. 2451–2454. DOI: 10.1109/iscas.2007.378616.

[15] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the State of Neural Network Pruning?" In *Proceedings of the 3rd MLSys Conference*, Austin, TX, USA, 2020, pp. 1–17.

[16] S. M. Bohte, "The Evidence for Neural Information Processing with Precise Spike-times: A Survey," *Natural Computing*, vol. 3, no. 2, pp. 195–206, 2004, ISSN: 15677818. DOI: 10.1023/B:NACO.0000027755.02868.60.

[17] J. M. Brader, W. Senn, and S. Fusi, "Learning Real-World Stimuli in a Neural Network With Spike-Driven Synaptic Dynamics," *Neural computation*, vol. 19, no. 11, pp. 2881–2912, 2007, ISSN: 0899-7667. DOI: 10.1162/neco.2007.19.11.2881.

[18] G. Bradski, *The OpenCV Library*, 2000.

[19] C. Brandli, T. A. Mantel, M. Hutter, M. A. Höpflinger, R. Berner, R. Siegwart, and T. Delbruck, "Adaptive Pulsed Laser Line Extraction for Terrain Reconstruction Using a Dynamic Vision Sensor," *Frontiers in Neuroscience*, vol. 7, no. 8 JAN, pp. 1–9, 2014, ISSN: 1662453X. DOI: 10.3389/fnins.2013.00275.

[20] A. N. Burkitt, "A Review of the Integrate-and-Fire Neuron Model: I. Homogeneous Synaptic Input," *Biological Cybernetics*, vol. 95, no. 1, pp. 1–19, 2006, ISSN: 03401200. DOI: 10.1007/s00422-006-0068-6.

[21] V. Campos, B. Jou, X. Giro-i-Nieto, J. Torres, and S.-F. Chang, "Skip RNN: Learning to Skip State Updates in Recurrent Neural Networks," in *International Conference on Learning Representations*, 2018, pp. 1–17, ISBN: 9783901608353. DOI: 10.1021/acs.jcim.6b00754. [Online]. Available: http://arxiv.org/abs/1708.06834.

[22] Y. Cao, Y. Chen, and D. Khosla, "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015, ISSN: 15731405. DOI: 10.1007/s11263-014-0788-3. [Online]. Available: http://link.springer.com/10.1007/s11263-014-0788-3.

[23] Z. Cao, T. Simon, S. E. Wei, and Y. Sheikh, "Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 1302–1310, 2017, ISSN: 10636919. DOI: 10.1109/CVPR.2017.143.

[24] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, A. Amir, D. B. D. Rubin, F. Akopyan, E. McQuinn, W. P. Risk, and D. S. Modha, "Cognitive Computing Building Block: A Versatile and Efficient Digital Neuron Model for Neurosynaptic Cores," in *Proceedings of the International Joint Conference on Neural Networks*, Dallas, TX, 2013, ISBN: 9781467361293. DOI: 10.1109/IJCNN.2013.6707077.

[25] L. Cavigelli and L. Benini, "CBinfer: Exploiting Frame-to-Frame Locality for Faster Convolutional Network Inference on Video Streams," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 5, pp. 1451–1465, May 2020, ISSN: 15582205. DOI: 10.1109/TCSVT.2019.2903421.

[26] F. S. Chance, J. B. Aimone, S. S. Musuvathy, M. R. Smith, C. M. Vineyard, and F. Wang, "Crossing the Cleft: Communication Challenges Between Neuroscience and Artificial Intelligence," *Frontiers in Computational Neuroscience*, vol. 14, p. 39, May 2020, ISSN: 16625188. DOI: 10.3389/fncom.2020.00039. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncom.2020.00039/full.

[27] R. Chen, H. Ma, P. Guo, S. Xie, P. Li, and D. Wang, "Low Latency Spiking ConvNets with Restricted Output Training and False Spike Inhibition," in *International Joint Conference on Neural Networks*, IEEE, 2018, pp. 404–411, ISBN: 9781509060146. DOI: 10.1109/IJCNN.2018.8489400.

[28] Y. H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016*, 2016, pp. 367–379, ISBN: 9781467389471. DOI: 10.1109/ISCA.2016.40.

[29] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," in *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, IEEE Computer Society, Jan. 2015, pp. 609–622, ISBN: 9781479969982. DOI: 10.1109/MICRO.2014.58.

[30] C. H. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," *Lecture Notes in Computer Science*, vol. 10482 LNCS, pp. 251–268, 2017, ISSN: 16113349. DOI: 10.1007/978-3-319-68167-2{_}18. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-68167-2_18.

[31] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient Primitives for Deep Learning," NVIDIA, Santa Clara, CA 95050, Tech. Rep., 2014. [Online]. Available: https://arxiv.org/pdf/1410.0759.pdf.

[32] F. Chollet, *Keras*, 2015. [Online]. Available: https://keras.io.

[33] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit Quantization of Neural Networks for Efficient Inference," in *Proceedings - 2019 International Conference on Computer Vision Workshop, ICCVW 2019*, 2019, pp. 3009–3018, ISBN: 9781728150239. DOI: 10.1109/ICCVW.2019.00363. [Online]. Available: http://arxiv.org/abs/1902.06822.

[34] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," in *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, Montreal, Canada, 2015, pp. 1–9. [Online]. Available: http://arxiv.org/abs/1511.00363.

[35] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," *arXiv:1602.02830*, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830.

[36] D. D. Cox and T. Dean, "Neural Networks and Neuroscience-Inspired Computer Vision," *Current Biology*, vol. 24, no. 18, R921–R929, 2014, ISSN: 09609822. DOI: 10.1016/j.cub.2014.08.026. [Online]. Available: http://dx.doi.org/10.1016/j.cub.2014.08.026.

[37] M. Davies, "Benchmarks for Progress in Neuromorphic Computing," *Nature Machine Intelligence*, vol. 1, no. 9, pp. 386–388, 2019, ISSN: 2522-5839. DOI: 10.1038/s42256-019-0097-1. [Online]. Available: https://doi.org/10.1038/s42256-019-0097-1.

[38] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018, ISSN: 02721732. DOI: 10.1109/MM.2018.112130359.

[39] T. Delbruck, "Frame-Free Dynamic Digital Vision," in *Intl. Symp. on Secure-Life Electronics, Advanced Electronics for Quality Life and Society*, 2008, pp. 21–26. DOI: http://dx.doi.org/10.5167/uzh-17620.

[40] L. Deng, Y. Wu, X. Hu, L. Liang, Y. Ding, G. Li, G. Zhao, P. Li, and Y. Xie, "Rethinking the Performance Comparison Between SNNS and ANNS," *Neural Networks*, vol. 121, pp. 294–307, Sep. 2020, ISSN: 18792782. DOI: 10.1016/j.neunet.2019.09.005. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0893608019302667.

[41] P. U. Diehl and M. Cook, "Unsupervised Learning of Digit Recognition Using Spike-Timing-Dependent Plasticity," *Frontiers in Computational Neuroscience*, vol. 9, no. August, pp. 1–9, 2015, ISSN: 1662-5188. DOI: 10.3389/fncom.2015.00099. [Online]. Available: http://journal.frontiersin.org/Article/10.3389/fncom.2015.00099/abstract.

[42] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. C. Liu, and M. Pfeiffer, "Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing," in *Proceedings of the International Joint Conference on Neural Networks*, Killarney, Ireland, 2015, ISBN: 9781479919604. DOI: 10.1109/IJCNN.2015.7280696.

[43] P. U. Diehl, B. U. Pedroni, A. Cassidy, P. Merolla, E. Neftci, and G. Zarrella, "TrueHappiness: Neuromorphic Emotion Recognition on TrueNorth," in *Proceedings of the International Joint Conference on Neural Networks*, Vancouver, Canada, 2016, pp. 4278–4285, ISBN: 9781509006199. DOI: 10.1109/IJCNN.2016.7727758.

[44] P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci, "Conversion of Artificial Recurrent Neural Networks to Spiking Neural Networks for Low-Power Neuromorphic Hardware," in *2016 IEEE International Conference on Rebooting Computing*, 2016, ISBN: 9781509013708. DOI: 10.1109/ICRC.2016.7738691.

[45] S. K. Esser, J. V. Arthur, P. A. Merolla, D. S. Modha, and R. Appuswamy, "Backpropagation for Energy-Efficient Neuromorphic Computing," in *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, Montreal, Canada, 2015, pp. 1–9. [Online]. Available: http://papers.nips.cc/paper/5862-backpropagation-for-energy-efficient-neuromorphic-computing.

[46] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. di Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, and D. S. Modha, "Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 113, no. 41, pp. 11 441–11 446, 2016, ISSN: 0027-8424. DOI: 10.1073/pnas.1604850113. [Online]. Available: http://arxiv.org/abs/1603.08270%0Ahttp://dx.doi.org/10.1073/pnas.1604850113.

[47] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-Level Classification of Skin Cancer With Deep Neural Networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017, ISSN: 1476-4687.

DOI: 10.1038/nature21056. [Online]. Available: https://doi.org/10.1038/nature21056.

[48] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, "Aggressive Quadrotor Flight Through Narrow Gaps With Onboard Sensing and Computing Using Active Vision," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5774–5781. DOI: 10.1109/ICRA.2017.7989679.

[49] C. Farabet, R. Paz, J. Pérez-Carrasco, C. Zamarreño-Ramos, A. Linares-Barranco, Y. LeCun, E. Culurciello, T. Serrano-Gotarredona, and B. Linares-Barranco, "Comparison between Frame-Constrained Fix-Pixel-Value and Frame-Free Spiking-Dynamic-Pixel ConvNets for Visual Processing," *Frontiers in Neuroscience*, vol. 6, no. April, pp. 1–12, 2012, ISSN: 1662-4548. DOI: 10.3389/fnins.2012.00032. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnins.2012.00032/abstract.

[50] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Convolutional Two-Stream Network Fusion for Video Action Recognition," no. i, 2016, ISSN: 10636919. DOI: 10.1109/CVPR.2016.213. [Online]. Available: http://arxiv.org/abs/1604.06573.

[51] M. Fischetti and J. Jo, "Deep Neural Networks as 0-1 Mixed Integer Linear Programs: A Feasibility Study," Department of Information Engineering, University of Padova, Tech. Rep., 2017. [Online]. Available: http://arxiv.org/abs/1712.06174.

[52] E. P. Frady, G. Orchard, D. Florey, N. Imam, R. Liu, J. Mishra, J. Tse, A. Wild, F. T. Sommer, and M. Davies, "Neuromorphic Nearest Neighbor Search Using Intel's Pohoiki Springs," in *ACM International Conference Proceeding Series*, 2020, pp. 1–9, ISBN: 9781450361231. DOI: 10.1145/3381755.3398695.

[53] K. Fukushima, "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980, ISSN: 1432-0770. DOI: 10.1007/BF00344251. [Online]. Available: https://doi.org/10.1007/BF00344251.

[54] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition," in *Artificial Neural Networks: Theoretical Concepts*, IEEE Computer Society Press, 1988, pp. 136–144.

[55] S. Furber, D. Lester, L. Plana, J. Garside, E. Painkras, S. Temple, and A. Brown, "Overview of the SpiNNaker System Architecture," *IEEE Trans. Comput.*, vol. 62, no. 12, pp. 2454–2467, 2013.

[56] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker Project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014, ISSN: 00189219. DOI: 10.1109/JPROC.2014.2304638.

[57] R. Gallager and D. van Voorhis, "Optimal Source Codes for Geometrically Distributed Integer Alphabets," *IEEE Transactions on Information Theory*, vol. 21, no. 2, pp. 228–230, Mar. 1975, ISSN: 1557-9654. DOI: 10.1109/TIT.1975.1055357.

[58] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based Vision: A Survey," pp. 1–25, 2019. [Online]. Available: http://arxiv.org/abs/1904.08405.

[59] V. C. Gaudet, "Low-Power Design Techniques for State-of-the-Art CMOS Technologies," in *Recent Progress in the Boolean Domain*, B. Steinbach, Ed., 1st ed., Newcastle upon Tyne, UK: Cambridge Scholars Publishing, 2014, ch. 4.1, pp. 187–212, ISBN: 978-1-4438-5638-6.

[60] A. Geiger, P. Lenz, and R. Urtasun, "Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[61] G. Georgiadis, "Accelerating Convolutional Neural Networks via Activation Map Compression," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7078–7088, ISBN: 9781728132938. DOI: 10.1109/CVPR.2019.00725. [Online]. Available: https://ai.intel.com/nervana-nnp/.

[62] W. Gerstner, R. Kempter, J. L. van Hemmen, and H. Wagner, "A Neuronal Learning Rule for Sub-Millisecond Temporal Coding," *Nature*, vol. 383, no. 6595, pp. 76–78, 1996. DOI: 10.1038/383076a0. [Online]. Available: http://infoscience.epfl.ch/record/97773.

[63] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014. [Online]. Available: neuronaldynamics.epfl.ch/online/index.html.

[64] W. Gerstner, M. Lehmann, V. Liakoni, D. Corneil, and J. Brea, "Eligibility Traces and Plasticity on Behavioral Time Scales: Experimental Support of NeoHebbian Three-Factor Learning Rules," *Frontiers in Neural Circuits*, vol. 12, no. July, pp. 1–16, 2018, ISSN: 16625110. DOI: 10.3389/fncir.2018.00053. [Online]. Available: http://arxiv.org/abs/1801.05219%0Ahttp://dx.doi.org/10.3389/fncir.2018.00053.

[65] W. Gerstner, R. Ritz, and J. L. van Hemmen, "Why Spikes? Hebbian Learning and Retrieval of Time-Resolved Excitation Patterns," *Biological Cybernetics*, vol. 69, no. 5-6, pp. 503–515, 1993, ISSN: 03401200. DOI: 10.1007/BF00199450.

[66] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, Columbus, Ohio, 2014, pp. 696–701, ISBN: 9781479943098. DOI: 10.1109/CVPRW.2014.106.

[67] Y. S. Goo, J. H. Ye, S. Lee, Y. Nam, S. B. Ryu, and K. H. Kim, "Retinal Ganglion Cell Responses to Voltage and Current Stimulation in Wild-Type and rd1 Mouse Retinas," *Journal of Neural Engineering*, vol. 8, no. 3, p. 035 003, Jun. 2011, ISSN: 1741-2560. DOI: 10.1088/1741-2560/8/3/035003. [Online]. Available: http://stacks.iop.org/1741-2552/8/i=3/a=035003?key=crossref.5a2ac3c65edf65bacbd887a07e100f34.

[68] D. A. Gudovskiy and L. Rigazio, "ShiftCNN: Generalized Low-Precision Architecture for Inference of Convolutional Neural Networks," *arXiv:1706.02393*, 2017. [Online]. Available: http://arxiv.org/abs/1706.02393.

[69] R. Gütig and H. Sompolinsky, "The Tempotron: A Neuron That Learns Spike Timing-Based Decisions," *Nature Neuroscience*, vol. 9, no. 3, pp. 420–428, 2006, ISSN: 10976256. DOI: 10.1038/nn1643. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/16474393.

[70] A. E. Hadjinicolaou, C. O. Savage, N. V. Apollo, D. J. Garrett, S. L. Cloherty, M. R. Ibbotson, and B. J. O'Brien, "Optimizing the Electrical Stimulation of Retinal Ganglion Cells," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 2015, ISSN: 15344320. DOI: 10.1109/TNSRE.2014.2361900.

[71] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in *ICLR*, 2015, pp. 1–13. DOI: abs/1510.00149/1510.00149. [Online]. Available: http://arxiv.org/abs/1510.00149.

[72] A. Hanuschkin, S. Kunkel, M. Helias, A. Morrison, and M. Diesmann, "A General and Efficient Method for Incorporating Precise Spike Times in Globally Time-driven Simulations.," *Frontiers in Neuroinformatics*, vol. 4, no. October, p. 113, 2010, ISSN: 1662-5196. DOI: 10.3389/fninf.2010.00113.

[73] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *CoRR*, vol. abs/1502.0, 2015, ISSN: 15505499. DOI: 10.1109/ICCV.2015.123. [Online]. Available: http://arxiv.org/abs/1502.01852.

[74] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778, ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.90.

[75] W. He, Y. Wu, L. Deng, G. Li, H. Wang, Y. Tian, W. Ding, W. Wang, and Y. Xie, "Comparing SNNs and RNNs on Neuromorphic Vision Datasets: Similarities and Differences," *arXiv:2005.02183*, 2020. [Online]. Available: http://arxiv.org/abs/2005.02183.

[76] D. O. Hebb, *The Organization of Behavior*. New York: Wiley & Sons, 1949.

[77] M. Horowitz, "Computing's Energy Problem (and What We Can Do About It)," in *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, vol. 57, San Francisco, CA, 2014, pp. 10–14, ISBN: 9781479909186. DOI: 10.1109/ISSCC.2014.6757323.

[78] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv:1704.04861*, 2017. DOI: arXiv:1704.04861. [Online]. Available: http://arxiv.org/abs/1704.04861.

[79] Y. Hu, H. Tang, Y. Wang, and G. Pan, "Spiking Deep Residual Network," *arXiv:1805.01352*, 2018. [Online]. Available: http://arxiv.org/abs/1805.01352.

[80] Y. Hu, H. Liu, M. Pfeiffer, and T. Delbruck, "DVS Benchmark Datasets for Object Tracking, Action Recognition, and Object Recognition," *Frontiers in Neuroscience*, vol. 10, 2016, ISSN: 1662453X. DOI: 10.3389/fnins.2016.00405.

[81] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *arXiv:1602.02830*, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830.

[82] E. Hunsberger and C. Eliasmith, "Spiking Deep Networks with LIF Neurons," *arXiv:1510.08829*, 2015. [Online]. Available: http://arxiv.org/abs/1510.08829.

[83] E. Hunsberger and C. Eliasmith, "Training Spiking Deep Networks for Neuromorphic Hardware," *arXiv:1611.05141*, 2016. DOI: 10.13140/RG.2.2.10967.06566. [Online]. Available: http://arxiv.org/abs/1611.05141%0Ahttp://dx.doi.org/10.13140/RG.2.2.10967.06566.

[84] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and \<0.5MB Model Size," *arXiv*, pp. 1–5, 2016, ISSN: 0302-9743. DOI: 10.1007/978-3-319-24553-9. [Online]. Available: http://arxiv.org/abs/1602.07360.

[85] M. Im and S. I. Fried, "Temporal Properties of Network-Mediated Responses to Repetitive Stimuli Are Dependent Upon Retinal Ganglion Cell Type," *Journal of Neural Engineering*, vol. 13, no. 2, pp. 1–26, 2016, ISSN: 17412552. DOI: 10.1088/1741-2560/13/2/025002.

[86] G. Indiveri, E. Chicca, and R. J. Douglas, "Artificial Cognitive Systems: From VLSI Networks of Spiking Neurons to Neuromorphic Cognition," *Cognitive Computation*, vol. 1, no. 2, pp. 119–127, 2009, ISSN: 1866-9964. DOI: 10.1007/s12559-008-9003-6. [Online]. Available: https://doi.org/10.1007/s12559-008-9003-6.

[87] G. Indiveri, F. Corradi, and N. Qiao, "Neuromorphic Architectures for Spiking Deep Neural Networks," in *Technical Digest - International Electron Devices Meeting, IEDM*, 2015, pp. 1–4, ISBN: 9781467398930. DOI: 10.1109/IEDM.2015.7409623.

[88] INIlabs, *AEDAT fileformat*, 2010. [Online]. Available: https://inilabs.com/support/software/fileformat/.

[89] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv:1502.03167*, 2015, ISSN: 0717-6163. DOI: 10.1007/s13398-014-0173-7.2. [Online]. Available: http://arxiv.org/abs/1502.03167.

[90] A. Jain, A. Phanishayee, J. Mars, L. Tang, and G. Pekhimenko, "GIST: Efficient Data Encoding for Deep Neural Network Training," in *Proceedings - International Symposium on Computer Architecture*, 2018, pp. 776–789, ISBN: 9781538659847. DOI: 10.1109/ISCA.2018.00070.

[91] A. Jalligampala, S. Sekhar, E. Zrenner, and D. L. Rathbun, "Optimal Voltage Stimulation Parameters for Network-Mediated Responses in Wild Type and rd10 Mouse Retinal Ganglion Cells," *Journal of Neural Engineering*, vol. 14, no. 2, p. 026 004, Apr. 2017, ISSN: 1741-2560. DOI: 10.1088/1741-2552/14/2/026004. [Online]. Available: http://stacks.iop.org/1741-2552/14/i=2/a=026004?key=crossref.0325bc303d881435c224b55873f2fc78.

[92] S. Jastrzebski, D. Arpit, N. Ballas, V. Verma, T. Che, and Y. Bengio, "Residual Connections Encourage Iterative Inference," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, no. 2016, pp. 1–14, 2018.

[93] L. H. Jepson, P. Hottowy, G. A. Weiner, W. Dabrowski, A. M. Litke, and E. Chichilnisky, "High-Fidelity Reproduction of Spatiotemporal Visual Signals for Retinal Prosthesis," *Neuron*, vol. 83, no. 1, pp. 87–92, Jul. 2014, ISSN: 0896-6273. DOI: 10.1016/J.NEURON.2014.04.044. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0896627314003596.

[94] S. Ji, M. Yang, K. Yu, and W. Xu, "3D Convolutional Neural Networks for Human Action Recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–31, 2013, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2012.59. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6165309%5Cnhttp://www.ncbi.nlm.nih.gov/pubmed/22392705.

[95] P. Judd, J. Albericio, and A. Moshovos, "Stripes: Bit-Serial Deep Neural Network Computing," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 80–83, 2017, ISSN: 15566056. DOI: 10.1109/LCA.2016.2597140.

[96] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)," *Frontiers in Neuroscience*, vol. 14, no. May, pp. 1–11, 2020, ISSN: 1662453X. DOI: 10.3389/fnins.2020.00424.

[97] M. Kaku, *The Future of the Mind: The Scientific Quest to Understand, Enhance, and Empower the Mind*, 1st ed. New York: Doubleday, Random House, 2014, pp. 1–377, ISBN: 038553082X.

[98] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. F. Li, "Large-Scale Video Classification With Convolutional Neural Networks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732, ISBN: 9781479951178. DOI: 10.1109/CVPR.2014.223.

[99] R. Kemker and C. Kanan, "FearNet: Brain-Inspired Model for Incremental Learning," in *ICLR 2018*, 2018, pp. 1–10.

[100] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-Based Spiking Deep Convolutional Neural Networks for Object Recognition," *Neural Networks*, vol. 99, pp. 56–67, 2018, ISSN: 18792782. DOI: 10.1016/j.neunet.2017.12.005. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0893608017302903%20https://doi.org/10.1016/j.neunet.2017.12.005.

[101] J. Kim, H. Kim, S. Huh, J. Lee, and K. Choi, "Deep Neural Networks With Weighted Spikes," *Neurocomputing*, vol. 311, pp. 373–386, 2018, ISSN: 18728286. DOI: 10.1016/j.neucom.2018.05.087.

[102] R. Kim, Y. Li, and T. J. Sejnowski, "Simple Framework for Constructing Functional Spiking Recurrent Neural Networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 116, no. 45, pp. 22 811–22 820, 2019, ISSN: 10916490. DOI: 10.1073/pnas.1905926116.

[103] S. Kim, S. Park, B. Na, and S. Yoon, "Spiking-YOLO: Spiking Neural Network for Real-time Object Detection," *arXiv:1903.06530*, 2019. [Online]. Available: https://arxiv.org/pdf/1903.06530.pdf.

[104] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming Catastrophic Forgetting in Neural Networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 114, no. 13, pp. 3521–3526, 2017, ISSN: 10916490. DOI: 10.1073/pnas.1611835114. [Online]. Available: http://arxiv.org/abs/1708.02072.

[105] I. Kiselev, D. Neil, and S. C. Liu, "Event-Driven Deep Neural Network Hardware System for Sensor Fusion," in *Proceedings - IEEE International Symposium on Circuits and Systems*, Montreal, Canada, 2016, pp. 2495–2498, ISBN: 9781479953400. DOI: 10.1109/ISCAS.2016.7539099.

[106] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," University of Toronto, Tech. Rep., 2009, pp. 1–60. DOI: 10.1.1.222.9220. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Learning+Multiple+Layers+of+Features+from+Tiny+Images#0.

[107] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, Lake Tahoe, NV, 2012, pp. 1–9, ISBN: 9781627480031.

[108] J. Kubilius, M. Schrimpf, K. Kar, H. Hong, N. J. Majaj, R. Rajalingham, E. B. Issa, P. Bashivan, J. Prescott-Roy, K. Schmidt, A. Nayebi, D. Bear, D. L. K. Yamins, and J. J. DiCarlo, "Brain-Like Object Recognition with High-Performing Shallow Recurrent ANNs," in *33rd Conference on Neural Information Processing Systems*, 2019. [Online]. Available: http://arxiv.org/abs/1909.06161.

[109] A. Kugele, T. Pfeil, M. Pfeiffer, and E. Chicca, "Efficient Processing of Spatio-Temporal Data Streams with Spiking Neural Networks," *Frontiers in Neuroscience*, vol. 14, no. May, p. 439, 2020, ISSN: 1662-453X. DOI: 10.3389/FNINS.2020.00439. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2020.00439/abstract.

[110] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman, "HOTS: A Hierarchy of Event-Based Time-Surfaces for Pattern Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 7, pp. 1346–1359, 2017, ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2574707.

[111] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989, ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.

[112] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, ISSN: 0028-0836. DOI: 10.1038/nature14539. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84930630277&partnerID=40&md5=befeefa64ddca265c713cf81f4e2fc54.

[113] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," in *Proceedings of the IEEE*, vol. 86, IEEE, 1998, pp. 2278–2323. DOI: 10.1109/5.726791.

[114] G.-H. Lee, D. Alvarez-Melis, and T. S. Jaakkola, "Towards Robust, Locally Linear Deep Networks," in *ICLR*, 2019, pp. 1–21.

[115] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training Deep Spiking Neural Networks using Backpropagation," *Frontiers in Neuroscience*, vol. 10, no. NOV, pp. 1–10, 2016, ISSN: 1662453X. DOI: 10.3389/fnins.2016.00508.

[116] S. W. Lee, D. K. Eddington, and S. I. Fried, "Responses to Pulsatile Subretinal Electric Stimulation: Effects of Amplitude and Duration," *Journal of Neurophysiology*, vol. 109, no. 7, pp. 1954–1968, 2013, ISSN: 00223077. DOI: 10.1152/jn.00293.2012.

[117] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 x 128 120dB 15us Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[118] C. K. Lin, A. Wild, G. N. Chinya, Y. Cao, M. Davies, D. M. Lavery, and H. Wang, "Programming Spiking Neural Networks on Intel's Loihi," *Computer*, vol. 51, no. 3, pp. 52–61, 2018, ISSN: 00189162. DOI: 10.1109/MC.2018.157113521.

[119] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime Neural Pruning," in *Advances in Neural Information Processing Systems*, 2017, pp. 2178–2188. [Online]. Available: https://papers.nips.cc/paper/6813-runtime-neural-pruning.

[120] S. C. Liu, B. Rueckauer, E. Ceolini, A. Huber, and T. Delbruck, "Event-Driven Sensing for Efficient Perception: Vision and Audition Algorithms," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 29–37, 2019, ISSN: 15580792. DOI: 10.1109/MSP.2019.2928127.

[121] S.-C. Liu, T. Delbruck, G. Indiveri, R. Douglas, and A. Whatley, *Event-Based Neuromorphic Systems*. Chichester, UK: John Wiley \& Sons, 2015, p. 440, ISBN: 0470018496. [Online]. Available: https://books.google.com/books?id=MuvsBQAAQBAJ&pgis=1.

[122] I. Lungu, F. Corradi, and T. Delbrück, "Live Demonstration: Convolutional Neural Network Driven by Dynamic Vision Sensor Playing RoShamBo," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, p. 1. DOI: 10.1109/ISCAS.2017.8050403.

[123] W. Maass, "Fast Sigmoidal Networks via Spiking Neurons," *Neural Computation*, vol. 9, no. 2, pp. 279–304, 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.2.279. [Online]. Available: http://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.2.279.

[124] W. Maass, "Networks of Spiking Neurons: The Third Generation of Neural Network Models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997, ISSN: 08936080. DOI: 10.1016/S0893-6080(97)00011-7.

[125] W. Maass, "Noisy Spiking Neurons with Temporal Coding have more Computational Power than Sigmoidal Neurons," in *Advances in Neural Information Processing Systems*, vol. 9, 1997, pp. 211–217. DOI: 10.1.1.49.2055.

[126] W. Maass, "To Spike or Not to Spike: That Is the Question," *Proceedings of the IEEE*, vol. 103, no. 12, pp. 2219–2224, 2015, ISSN: 00189219. DOI: 10.1109/JPROC.2015.2496679.

[127] W. Maass and H. Markram, "On the Computational Power of Circuits of Spiking Neurons," *Journal of Computer and System Sciences*, vol. 69, no. 4, pp. 593–616, 2004, ISSN: 00220000. DOI: 10.1016/j.jcss.2004.04.001. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0022000004000406.

[128] H. Martin and J. Conradt, "Spiking Neural Networks for Vision Tasks," 2015.

[129] S. Martinez-Conde, S. L. Macknik, and D. H. Hubel, "Microsaccadic Eye Movements and Firing of Single Cells in the Striate Cortex of Macaque Monkeys," *Nature Neuroscience*, vol. 3, no. 3, pp. 251–258, Mar. 2000, ISSN: 10976256. DOI: 10.1038/72961. [Online]. Available: http://neurosci.nature.com.

[130] T. Masquelier and S. J. Thorpe, "Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity," *PLoS Computational Biology*, vol. 3, no. 2, pp. 0247–0257, 2007, ISSN: 1553734X. DOI: 10.1371/journal.pcbi.0030031.

[131] R. Massa, A. Marchisio, M. Martina, and M. Shafique, "An Efficient Spiking Neural Network for Recognizing Gestures with a DVS Camera on the Loihi Neuromorphic Processor," in *2020 International Joint Conference on Neural Networks (IJCNN)*, Glasgow, Scotland, 2020. [Online]. Available: http://arxiv.org/abs/2006.09985.

[132] N. Y. Masse, G. D. Grant, and D. J. Freedman, "Alleviating Catastrophic Forgetting Using Context-Dependent Gating and Synaptic Stabilization," *Proceedings of the National Academy of Sciences*, vol. 115, no. 44, E10467–E10475, 2018, ISSN: 0027-8424. DOI: 10.1073/pnas.1803839115. [Online]. Available: https://www.pnas.org/content/115/44/E10467.

[133] M. McGill and P. Perona, "Deciding How to Decide: Dynamic Routing in Artificial Neural Networks," in *International Conference on Machine Learning*, Sydney, Australia, Mar. 2017. [Online]. Available: http://arxiv.org/abs/1703.06217.

[134] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A Million Spiking-Neuron Integrated Circuit With a Scalable Communication Network and Interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014, ISSN: 0036-8075. DOI: 10.1126/science.1254642. [Online]. Available: http://www.sciencemag.org/cgi/doi/10.1126/science.1254642.

[135] N. Messikommer, D. Gehrig, A. Loquercio, and D. Scaramuzza, "Event-based Asynchronous Sparse Convolutional Networks," *arXiv:2003.09148v1*, 2020. [Online]. Available: https://youtu.be/LauQ6LWTkxM?t=4.

[136] J. W. Mink, R. J. Blumenschine, and D. B. Adams, "Ratio of Central Nervous System to Body Metabolism in Vertebrates: Its Constancy and Functional Basis," eng, *The American journal of physiology*, vol. 241, no. 3, pp. 203–12, Sep. 1981, ISSN: 0002-9513 (Print). DOI: 10.1152/ajpregu.1981.241.3.R203.

[137] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, "WRPN : Wide Reduced-Precision Networks," in *International Conference on Learning Representations*, 2018, pp. 1–11.

[138] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional Neural Networks using Logarithmic Data Representation," *arXiv*, 2016. [Online]. Available: http://arxiv.org/abs/1603.01025.

[139] D. P. Moeys, F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr, and T. Delbruck, "Steering a Predator Robot Using a Mixed Frame/Event-Driven Convolutional Neural Network," in *2016 2nd International Conference on Event-Based Control, Communication, and Signal Processing, EBCCSP 2016 - Proceedings*, Krakow, 2016, pp. 1–8, ISBN: 9781509041961. DOI: 10.1109/EBCCSP.2016.7605233. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7605233&isnumber=7605073.

[140] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, "Explaining Nonlinear Classification Decisions With Deep Taylor Decomposition," *Pattern Recognition*, vol. 65, pp. 211–222, 2017, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2016.11.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320316303582.

[141] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio, "On the Number of Linear Regions of Deep Neural Networks," in *NIPS*, 2014, pp. 1–9. DOI: 10.1007/978-1-4471-5779-3{_}4. [Online]. Available: http://arxiv.org/abs/1402.1869.

[142] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-Efficient ConvNets Through Approximate Computing," in *2016 IEEE Winter Conference on Applications of Computer Vision, WACV 2016*, 2016, ISBN: 9781509006410. DOI: 10.1109/WACV.2016.7477614.

[143] H. Mostafa, "Supervised Learning Based on Temporal Coding in Spiking Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 3227–3235, 2018, ISSN: 21622388. DOI: 10.1109/TNNLS.2017.2726060.

[144] H. Mostafa, B. U. Pedroni, S. Sheik, and G. Cauwenberghs, "Fast Classification Using Sparsely Active Spiking Networks," in *Proceedings - IEEE International Symposium on Circuits and Systems*, 2017, ISBN: 9781467368520. DOI: 10.1109/ISCAS.2017.8050527.

[145] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, "The Event-Camera Dataset and Simulator: Event-Based Data for Pose Estimation, Visual Odometry, and SLAM," *International Journal of Robotics Research*, vol. 36, no. 2, pp. 142–149, 2017, ISSN: 17413176. DOI: 10.1177/0278364917691115.

[146] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-Driven Random Back-Propagation: Enabling Neuromorphic Deep Learning Machines," *Frontiers in Neuroscience*, vol. 11, 2017, ISSN: 1662453X. DOI: 10.3389/fnins.2017.00324.

[147] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-based optimization to spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, 2019, ISSN: 15580792. DOI: 10.1109/MSP.2019.2931595.

[148] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, "Event-driven contrastive divergence for spiking neuromorphic systems," *Frontiers in Neuroscience*, vol. 7, no. January, pp. 1–14, 2014, ISSN: 1662-453X. DOI: 10.3389/fnins.2013.00272. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnins.2013.00272/abstract.

[149] D. Neil, J. H. Lee, T. Delbruck, and S.-C. Liu, "Delta Networks for Optimized Recurrent Network Computation," in *PMLR*, 2017. [Online]. Available: http://arxiv.org/abs/1612.05571.

[150] D. Neil and S. C. Liu, "Minitaur, an Event-Driven FPGA-Based Spiking Network Accelerator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 12, pp. 2621–2628, 2014, ISSN: 10638210. DOI: 10.1109/TVLSI.2013.2294916.

[151] D. Neil, M. Pfeiffer, and S. C. Liu, "Learning to be Efficient: Algorithms for Training Low-Latency, Low-Compute Deep Spiking Neural Networks," in *Proceedings of the ACM Symposium on Applied Computing*, Pisa, Italy, 2016, pp. 293–298, ISBN: 9781450337397. DOI: 10.1145/2851613.2851724. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2851613.2851724.

[152] B. Nessler, W. Maass, and M. Pfeiffer, "STDP Enables Spiking Neurons to Detect Hidden Causes of Their Inputs," in *Advances in Neural Information Processing Systems*, vol. 22, Vancouver, Canada, 2009, pp. 1357–1365, ISBN: 9781615679119.

[153] J. Y. H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond Short Snippets: Deep Networks for Video Classification," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4694–4702, ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7299101.

[154] R. Nusselder, "Spike-Based Long Short-Term Memory Networks," PhD thesis, 2017.

[155] P. O'Connor, D. Neil, S. C. Liu, T. Delbruck, and M. Pfeiffer, "Real-Time Classification and Sensor Fusion With a Spiking Deep Belief Network," *Frontiers in Neuroscience*, no. 7 OCT, 2013, ISSN: 16624548. DOI: 10.3389/fnins.2013.00178.

[156] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades," *Frontiers in Neuroscience*, 2015.

[157] G. Orchard, X. Lagorce, C. Posch, S. B. Furber, R. Benosman, and F. Galluppi, "Real-Time Event-Driven Spiking Neural Network Object Recognition on the SpiNNaker Platform," in *Proceedings - IEEE International Symposium on Circuits and Systems*, Lisbon, Portugal, 2015, pp. 2413–2416, ISBN: 9781479983919. DOI: 10.1109/ISCAS.2015.7169171.

[158] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman, "HFirst: A Temporal Approach to Object Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 10, pp. 2028–2040, Oct. 2015, ISSN: 01628828. DOI: 10.1109/TPAMI.2015.2392947.

[159] E. Park, J. Ahn, and S. Yoo, "Weighted-Entropy-Based Quantization for Deep Neural Networks," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, IEEE, Nov. 2017, pp. 7197–7205, ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.761.

[160] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[161] A. Patiño-Saucedo, H. Rostro-Gonzalez, T. Serrano-Gotarredona, and B. Linares-Barranco, "Event-Driven Implementation of Deep Spiking Convolutional Neural Networks for Supervised Classification Using the SpiNNaker Neuromorphic Platform," *Neural Networks*, vol. 121, pp. 319–328, 2020, ISSN: 18792782. DOI: 10.1016/j.neunet.2019.09.008. [Online]. Available: https://doi.org/10.1016/j.neunet.2019.09.008.

[162] B. U. Pedroni, S. Das, J. V. Arthur, P. A. Merolla, B. L. Jackson, D. S. Modha, K. Kreutz-Delgado, and G. Cauwenberghs, "Mapping Generative Models onto a Network of Digital Spiking Neurons," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 10, no. 4, pp. 837–854, 2016, ISSN: 19324545. DOI: 10.1109/TBCAS.2016.2539352.

[163] C. Pehlevan, "A Spiking Neural Network with Local Learning Rules Derived From Nonnegative Similarity Matching," *arXiv:1902.01429*, 2019. [Online]. Available: http://arxiv.org/abs/1902.01429.

[164] J. A. Perez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, and B. Linares-Barranco, "Mapping from Frame-Driven to Frame-Free Event-Driven Vision Systems by Low-Rate Rate Coding and Coincidence Processing - Application to Feedforward ConvNets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2706–2719, 2013, ISSN: 01628828. DOI: 10.1109/TPAMI.2013.71.

[165] M. Pfeiffer and T. Pfeil, "Deep Learning With Spiking Neurons: Opportunities and Challenges," *Frontiers in Neuroscience*, vol. 12, no. October, 2018, ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00774. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00774/full.

[166] C. Posch, D. Matolin, and R. Wohlgenannt, "A QVGA 143 dB Dynamic Range Frame-Free PWM Image Sensor With Lossless Pixel-Level Video Compression and Time-Domain CDS," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, Jan. 2011, ISSN: 1558-173X. DOI: 10.1109/JSSC.2010.2085952.

[167] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, "Retinomorphic Event-Based Vision Sensors : Bioinspired Cameras With Spiking Output," *Proceedings of the IEEE*, vol. 102, no. 10, pp. 1470–1484, 2014, ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2346153.

[168] I. Pozzi, R. Nusselder, D. Zambrano, and S. Bohté, "Gating Sensory Noise in a Spiking Subtractive LSTM," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11139 LNCS, 2018, pp. 284–293, ISBN: 9783030014179. DOI: 10.1007/978-3-030-01418-6{_}28.

[169] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A Reconfigurable On-Line Learning Spiking Neuromorphic Processor Comprising 256 Neurons and 128K Synapses," *Frontiers in Neuroscience*, vol. 9, no. APR, pp. 1–17, 2015, ISSN: 1662453X. DOI: 10.3389/fnins.2015.00141.

[170] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the Expressive Power of Deep Neural Networks," in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., International Convention Centre, Sydney, Australia: PMLR, 2017, pp. 2847–2854. [Online]. Available: http://proceedings.mlr.press/v70/raghu17a.html.

[171] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 525–542, ISBN: 978-3-319-46493-0.

[172] N. Rathi, G. Srinivasan, P. Panda, and K. Roy, "Enabling Deep Spiking Neural Networks with Hybrid Conversion and Spike Timing Dependent Backpropagation," in *International Conference on Learning Representations*, 2020. [Online]. Available: http://arxiv.org/abs/1910.13141.

[173] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *Cvpr 2016*, pp. 779–788, 2016, ISSN: 10636919. DOI: 10.1016/j.nima.2015.05.028.

[174] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and Benchmarking of Machine Learning Accelerators," in *2019 IEEE High Performance Extreme Computing Conference, HPEC 2019*, IEEE, 2019, pp. 1–9, ISBN: 9781728150208. DOI: 10.1109/HPEC.2019.8916327.

[175] O. Rhodes, P. A. Bogdan, C. Brenninkmeijer, S. Davidson, D. Fellows, A. Gait, D. R. Lester, M. Mikaitis, L. A. Plana, A. G. Rowley, A. B. Stokes, and S. B. Furber, "Spynnaker: A Software Package for Running Pynn Simulations on Spinnaker," *Frontiers in Neuroscience*, vol. 12, no. November, 2018, ISSN: 1662453X. DOI: 10.3389/fnins.2018.00816.

[176] M. Riesenhuber and T. Poggio, "Hierarchical Models of Object Recognition in Cortex," *Nature neuroscience*, vol. 2, pp. 1019–1025, 1999. DOI: 10.1038/14819.

[177] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive Auto-Encoders: Explicit Invariance During Feature Extraction," in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, 2011, pp. 833–840, ISBN: 9781450306195.

[178] R. V. Rikhye, A. Gilra, and M. M. Halassa, "Thalamic Regulation of Switching Between Cortical Representations Enables Cognitive Flexibility," *Nature Neuroscience*, vol. 21, no. 12, pp. 1753–1763, 2018, ISSN: 1546-1726. DOI: 10.1038/s41593-018-0269-z. [Online]. Available: https://doi.org/10.1038/s41593-018-0269-z.

[179] R. W. Rodieck, *The First Steps in Seeing*. Oxford University Press, 1998.

[180] F. Rosenblatt, "A Comparison of Several Perceptron Models," in *Self-Organizing Systems*, M. C. Yovits, G. T. Jacobi, and G. D. Goldstein, Eds., Spartan Books, 1962. DOI: 10.1137/1006025. [Online]. Available: https://doi.org/10.1137/1006025.

[181] A. G. D. Rowley, C. Brenninkmeijer, S. Davidson, D. Fellows, A. Gait, D. R. Lester, L. A. Plana, O. Rhodes, A. B. Stokes, and S. B. Furber, "SpiNNTools: The Execution Engine for the SpiNNaker Platform," *Frontiers in Neuroscience*, vol. 13, Mar. 2019, ISSN: 1662-453X. DOI: 10.3389/fnins.2019.00231. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2019.00231/full.

[182] B. Rueckauer, *Spiking Neural Network Conversion Toolbox*, 2017. [Online]. Available: http://snntoolbox.readthedocs.io.

[183] B. Rueckauer and T. Delbruck, "Evaluation of Event-Based Algorithms for Optical Flow with Ground-Truth from Inertial Measurement Sensor," *Frontiers in Neuroscience*, vol. 10, no. APR, pp. 1–17, 2016, ISSN: 1662453X. DOI: 10.3389/fnins.2016.00176.

[184] B. Rueckauer, N. Känzig, S.-C. Liu, T. Delbruck, and Y. Sandamirskaya, "Closing the Accuracy Gap in an Event-Based Visual Recognition Task," Tech. Rep., May 2019. [Online]. Available: http://arxiv.org/abs/1906.08859.

[185] B. Rueckauer and S. C. Liu, "Conversion of Analog to Spiking Neural Networks using Sparse Temporal Coding," in *Proceedings - IEEE International Symposium on Circuits and Systems*, Florence, Italy: IEEE, 2018, pp. 8–12, ISBN: 9781538648810. DOI: 10.1109/ISCAS.2018.8351295. [Online]. Available: https://ieeexplore.ieee.org/document/8351295/%20http://snntoolbox.readthedocs.io.

[186] B. Rueckauer and S. C. Liu, "Linear Approximation of Deep Neural Networks for Efficient Inference on Video Data," in *European Signal Processing Conference*, 2019, ISBN: 9789082797039. DOI: 10.23919/EUSIPCO.2019.8902997.

[187] B. Rueckauer, I.-A. Lungu, Y. Hu, and M. Pfeiffer, "Theory and Tools for the Conversion of Analog to Spiking Convolutional Neural Networks," *arXiv:1612.04052*, 2016. [Online]. Available: http://arxiv.org/abs/1612.04052.

[188] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification," *Frontiers in Neuroscience*, vol. 11, no. December, pp. 1–12, 2017, ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00682. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnins.2017.00682/full.

[189] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986, ISSN: 1476-4687. DOI: 10.1038/323533a0. [Online]. Available: https://doi.org/10.1038/323533a0.

[190] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015, ISSN: 15731405. DOI: 10.1007/s11263-015-0816-y. [Online]. Available: http://dx.doi.org/10.1007/s11263-015-0816-y.

[191] S. B. Ryu, J. W. Choi, K. N. Ahn, Y. S. Goo, and K. H. Kim, "Amplitude Modulation-based Electrical Stimulation for Encoding Multipixel Spatiotemporal Visual Information in Retinal Neural Activities," *Journal of Korean Medical Science*, vol. 32, no. 6, p. 900, 2017, ISSN: 1011-8934. DOI: 10.3346/jkms.2017.32.6.900. [Online]. Available: https://synapse.koreamed.org/DOIx.php?id=10.3346/jkms.2017.32.6.900.

[192] J. Sawada, F. Akopyan, A. S. Cassidy, B. Taba, M. V. Debole, P. Datta, R. Alvarez-Icaza, A. Amir, J. V. Arthur, A. Andreopoulos, R. Appuswamy, H. Baier, D. Barch, D. J. Berg, C. d. Nolfo, S. K. Esser, M. Flickner, T. A. Horvath, B. L. Jackson, J. Kusnitz, S. Lekuch, M. Mastro, T. Melano, P. A. Merolla, S. E. Millman, T. K. Nayak, N. Pass, H. E. Penner, W. P. Risk, K. Schleupen, B. Shaw, H. Wu, B. Giera, A. T. Moody, N. Mundhenk, B. C. Van Essen, E. X. Wang, D. P. Widemann, Q. Wu, W. E. Murphy, J. K. Infantolino, J. A. Ross, D. R. Shires, M. M. Vindiola, R. Namburu, and D. S. Modha, "Truenorth Ecosystem for Brain-inspired Computing: Scalable Systems, Software, and Applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Piscataway, NJ, USA: IEEE Press, 2016, pp. 1–12, ISBN: 978-1-4673-8815-3. [Online]. Available: http://dl.acm.org/citation.cfm?id=3014904.3014920.

[193] M. Schaffner, F. K. Gurkaynak, A. Smolic, and L. Benini, "DRAM or No-DRAM? Exploring Linear Solver Architectures for Image Domain Warping in 28 Nm CMOS," in *Proceedings -Design, Automation and Test in Europe, DATE*, Institute of Electrical and Electronics Engineers Inc., Apr. 2015, pp. 707–712, ISBN: 9783981537048. DOI: 10.7873/date.2015.0408.

[194] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pp. 85–117, 2014, ISSN: 08936080. DOI: 10.1016/j.neunet.2014.09.003. [Online]. Available: http://arxiv.org/abs/1404.7828%20http://dx.doi.org/10.1016/j.neunet.2014.09.003.

[195] S. Schmitt, J. Klahn, G. Bellec, A. Grubl, M. Guttler, A. Hartel, S. Hartmann, D. Husmann, K. Husmann, S. Jeltsch, V. Karasenko, M. Kleider, C. Koke, A. Kononov, C. Mauch, E. Muller, P. Muller, J. Partzsch, M. A. Petrovici, S. Schiefer, S. Scholze, V. Thanasoulis, B. Vogginger, R. Legenstein, W. Maass, C. Mayr, R. Schuffny, J. Schemmel, and K. Meier, "Neuromorphic Hardware in the Loop: Training a Deep Spiking Network on the BrainScaleS Wafer-Scale System," in *Proceedings of the International Joint Conference on Neural Networks*, 2017, pp. 2227–2234, ISBN: 9781509061815. DOI: 10.1109/IJCNN.2017.7966125. [Online]. Available: http://arxiv.org/abs/1703.01909.

[196] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going Deeper in Spiking Neural Networks: VGG and Residual Architectures," *Frontiers in Neuroscience*, vol. 13, no. March, pp. 1–10, 2019. DOI: 10.3389/fnins.2019.00095. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2019.00095/full.

[197] T. Serra, C. Tjandraatmadja, and S. Ramalingam, "Bounding and Counting Linear Regions of Deep Neural Networks," *35th International Conference on Machine Learning, ICML 2018*, vol. 10, pp. 7243–7261, 2018.

[198] T. Serrano-Gotarredona, B. Linares-Barranco, F. Galluppi, L. Plana, and S. Furber, "ConvNets Experiments on SpiNNaker," in *Proceedings - IEEE International Symposium on Circuits and Systems*, Lisbon, Portugal, 2015, pp. 2405–2408, ISBN: 9781479983919. DOI: 10.1109/ISCAS.2015.7169169.

[199] S. B. Shrestha and G. Orchard, "Slayer: Spike Layer Error Reassignment in Time," in *Advances in Neural Information Processing Systems*, 2018, pp. 1412–1421.

[200] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps," in *2nd International Conference on Learning Representations, ICLR 2014 - Workshop Track Proceedings*, 2014.

[201] K. Simonyan and A. Zisserman, "Two-Stream Convolutional Networks for Action Recognition in Videos," in *Advances in Neural Information Processing Systems*, 2014, pp. 1–11, ISBN: 9788578110796. DOI: 10.1017/CBO9781107415324.004. [Online]. Available: http://arxiv.org/abs/1406.2199.

[202] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *ICLR*, Banff, Canada, 2014, pp. 1–14. DOI: 10.1016/j.infsof.2008.09.005. [Online]. Available: http://arxiv.org/abs/1409.1556.

[203] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman, "HATS: Histograms of Averaged Time Surfaces for Robust Event-Based Object Classification," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, no. Figure 1, pp. 1731–1740, 2018, ISSN: 10636919. DOI: 10.1109/CVPR.2018.00186.

[204] M. Sorbaro, Q. Liu, M. Bortone, and S. Sheik, "Optimizing the Energy Consumption of Spiking Neural Networks for Neuromorphic Applications," *Frontiers in Neuroscience*, vol. 14, p. 662, 2020, ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00662. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2020.00662.

[205] C. Stöckl and W. Maass, "Recognizing Images With at Most One Spike per Neuron," *arxiv:2001.01682*, 2019. [Online]. Available: http://arxiv.org/abs/2001.01682.

[206] C. Stöckl and W. Maass, "Classifying Images with Few Spikes per Neuron," *arXiv:2002.00860*, 2020. [Online]. Available: http://arxiv.org/abs/2002.00860.

[207] E. Stromatias, F. Galluppi, C. Patterson, and S. Furber, "Power Analysis of Large-Scale, Real-Time Neural Networks on SpiNNaker," in *Proceedings of the International Joint Conference on Neural Networks*, 2013, ISBN: 9781467361293. DOI: 10.1109/IJCNN.2013.6706927.

[208] E. Stromatias, D. Neil, F. Galluppi, M. Pfeiffer, S. C. Liu, and S. Furber, "Scalable Energy-Efficient, Low-Latency Implementations of Trained Spiking Deep Belief Networks on SpiNNaker," in *Proceedings of the International Joint Conference on Neural Networks*, Killarney, Ireland, 2015, pp. 1–8, ISBN: 9781479919604. DOI: 10.1109/IJCNN.2015.7280625.

[209] E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco, "An Event-Driven Classifier for Spiking Neural Networks Fed with Synthetic or Dynamic Vision Sensor Data," *Frontiers in Neuroscience*, vol. 11, no. June, pp. 1–17, 2017, ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00350. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnins.2017.00350/full.

[210] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, San Francisco, CA, 2017, pp. 4278–4284. DOI: 10.1016/j.patrec.2014.01.008. [Online]. Available: http://arxiv.org/abs/1602.07261.

[211] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, Massachusetts, 2015, pp. 1–9, ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298594. [Online]. Available: http://arxiv.org/abs/1409.4842v1.

[212] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas, Nevada, 2016, ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.308. [Online]. Available: http://arxiv.org/abs/1512.00567.

[213] S. Thorpe, A. Delorme, and R. Van Rullen, "Spike-Based Strategies for Rapid Processing," *Neural Networks*, vol. 14, no. 6-7, pp. 715–725, 2001, ISSN: 08936080. DOI: 10.1016/S0893-6080(01)00083-1.

[214] R. VanRullen and S. J. Thorpe, "Rate Coding Versus Temporal Order Coding: What the Retinal Ganglion Cells Tell the Visual Cortex," *Neural computation*, vol. 13, no. 6, pp. 1255–83, 2001, ISSN: 0899-7667. DOI: 10.1162/08997660152002852. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/11387046.

[215] J. D. Victor, "How the Brain Uses Time to Represent and Process Visual Information," *Brain Research*, vol. 886, no. 1-2, pp. 33–46, 2000, ISSN: 00068993. DOI: 10.1016/S0006-8993(00)02751-7.

[216] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware Automated Quantization with Mixed Precision," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8604–8612, ISBN: 9781728132938. DOI: 10.1109/CVPR.2019.00881.

[217] L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon, "Towards Fast Computation of Certified Robustness for ReLU Networks," in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 5276–5285. [Online]. Available: http://proceedings.mlr.press/v80/weng18a.html.

[218] J. Wu, C. Xu, D. Zhou, H. Li, and K. C. Tan, "Progressive Tandem Learning for Pattern Recognition with Deep Spiking Neural Networks," *arXiv:2007.01204*, 2020. [Online]. Available: http://arxiv.org/abs/2007.01204.

[219] A. Yousefzadeh, T. Serrano-Gotarredona, B. Linares-Barranco, M. A. Khoei, S. Hosseini, P. Holanda, S. Leroux, O. Moreira, J. Tapson, B. Dhoedt, and P. Simoens, "Asynchronous Spiking Neurons, the Natural Key to Exploit Temporal Sparsity," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 668–678, Dec. 2019, ISSN: 21563365. DOI: 10.1109/JETCAS.2019.2951121. [Online]. Available: https://ieeexplore.ieee.org/document/8890681/.

[220] Q. Yu, C. Ma, S. Song, G. Zhang, J. Dang, and K. C. Tan, "Constructing Accurate and Efficient Deep Spiking Neural Networks with Double-threshold and Augmented Schemes," *arXiv:2005.03231*, 2020. [Online]. Available: http://arxiv.org/abs/2005.03231.

[221] D. Zambrano and S. M. Bohte, "Fast and Efficient Asynchronous Neural Computation with Adapting Spiking Neural Networks," *arXiv:1609.02053*, 2016. [Online]. Available: http://arxiv.org/abs/1609.02053.

[222] D. Zambrano, R. Nusselder, H. S. Scholte, and S. Bohte, "Efficient Computation in Adaptive Artificial Spiking Neural Networks," *Frontiers in Neuroscience*, vol. 12, no. January, pp. 1–11, 2019, ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00987. [Online]. Available: http://arxiv.org/abs/1710.04838.

[223] L. Zhang, S. Zhou, T. Zhi, Z. Du, and Y. Chen, "TDSNN: From Deep Neural Networks to Deep Spike Neural Networks with Temporal-Coding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1319–1326. DOI: 10.1609/aaai.v33i01.33011319.

[224] M. Zhang, N. Zheng, D. Ma, and Z. Gu, "Efficient Spiking Neural Networks with Logarithmic Temporal Coding," *arXiv:1811.04233v1*, 2018.

[225] B. Zhao, R. Ding, S. Chen, B. Linares-Barranco, and H. Tang, "Feedforward Categorization on AER Motion Events Using Cortex-Like Features in a Spiking Neural Network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 9, pp. 1963–1978, 2015, ISSN: 21622388. DOI: 10.1109/TNNLS.2014.2362542.

[226] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights," in *International Conference on Learning Representations*, 2017. [Online]. Available: http://arxiv.org/abs/1702.03044.

[227] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," *arXiv:1606.06160*, 2016. [Online]. Available: http://arxiv.org/abs/1606.06160.

[228] X. Zhu, Y. Xiong, J. Dai, L. Yuan, and Y. Wei, "Deep Feature Flow for Video Recognition," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 4141–4150, 2017, ISSN: 15505499. DOI: 10.1109/CVPR.2017.441.

# CURRICULUM VITAE


## BODO RUECKAUER

Born on May 4, 1988
Citizen of Germany and Switzerland


### Education

| | |
|---|---|
| 2016-2020 | PhD in Computational Neuroscience, University of Zurich |
| | Thesis: *Event-Based Vision Processing in Deep Neural Networks* |
| | Supervision: Shih-Chii Liu, Tobi Delbruck, and Giacomo Indiveri |
| 2014-2016 | MSc in Physics, ETH Zurich |
| | Thesis: *Development of event-based algorithms for optical flow* |
| | Supervision: Tobi Delbruck |
| 2009-2014 | BSc in Physics, ETH Zurich |
| | Thesis: *Phase space analysis of an adaptive eIF neuron on VLSI silicon* |
| | Supervision: Giacomo Indiveri |

### Research Experience

| | |
|---|---|
| Oct 2019 - Jan 2020 | Visiting Research Fellowship, Chungbuk National University<br>*Interfacing event-based vision sensors with microelectrode arrays for retinal stimulation in vitro.* |
| Apr 2019 - Sep 2019 | Research Internship, Intel Labs Oregon<br>*Implementation of a compiler for deep neural networks on Intel's neuromorphic processor Loihi.* |
| Feb 2019 - May 2019 | Freelance Consulting, Huawei Technologies |

### Awards and Grants

| | |
|---|---|
| Feb 2019 | Korean-Swiss Young Researchers Exchange Program |
| | Grant of USD 10'000 for a 3-month visit. |
| Jul 2018 | Best New Neuromorph award |
| | Telluride Neuromorphic Cognition Engineering Workshop |
| Mar 2018 | Travel grant, Neuroscience Center Zurich |